Introduction to S-PLUS 2000 Level II

Fairouz Makhlouf

Statistical Support Staff

Center for Information Technology

National Institutes of Health

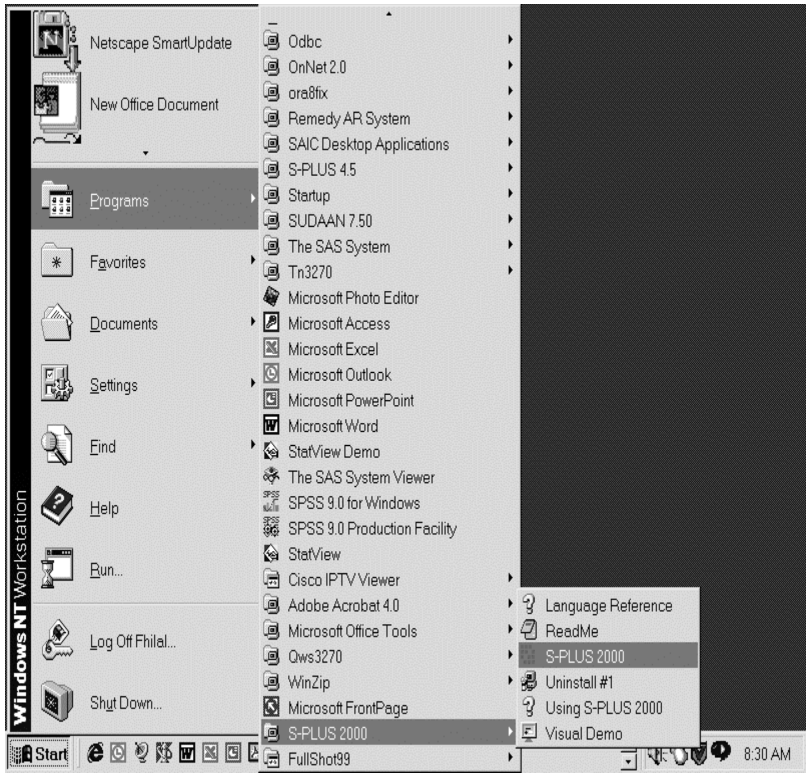
Contents:

INTRODUCTION

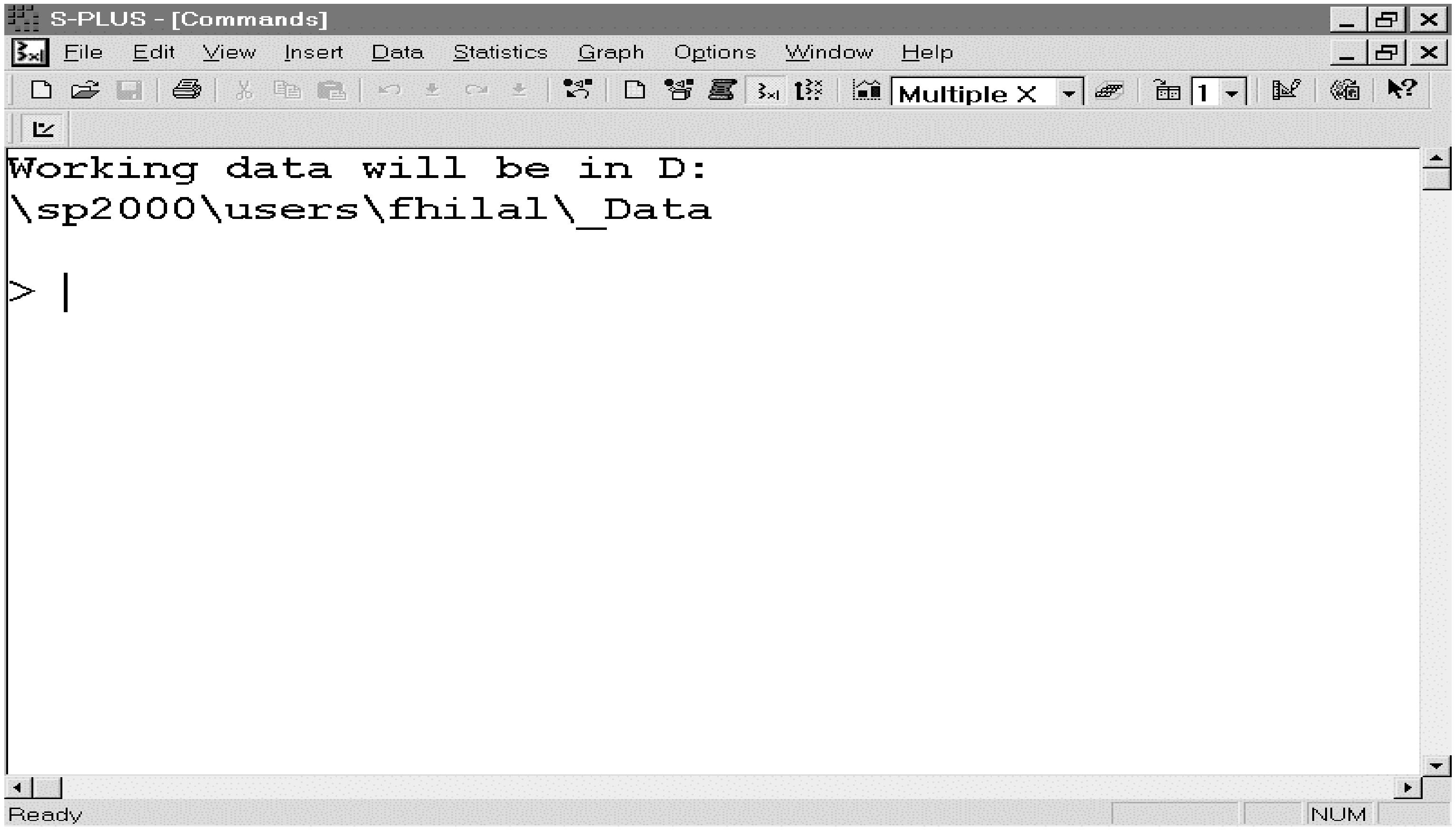
Contents:

Starting S-PLUS

- To Start S-Plus for Windows:
  - Go to Start button
  - Go to programs
  - Choose S-Plus 2000



- Wait for a copy-right message, followed by the S-Plus prompt.


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Multiple X
1
Ready
NUM

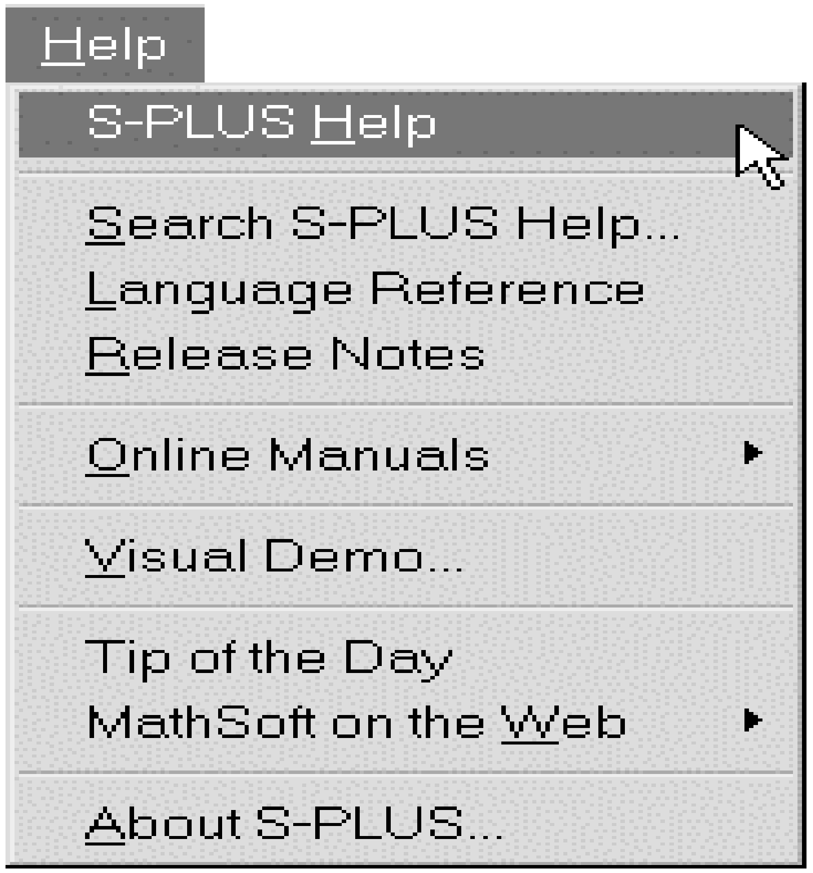
```
Working data will be in D:
\sp2000\users\fhilal\_Data

> |
```

Quitting S-PLUS

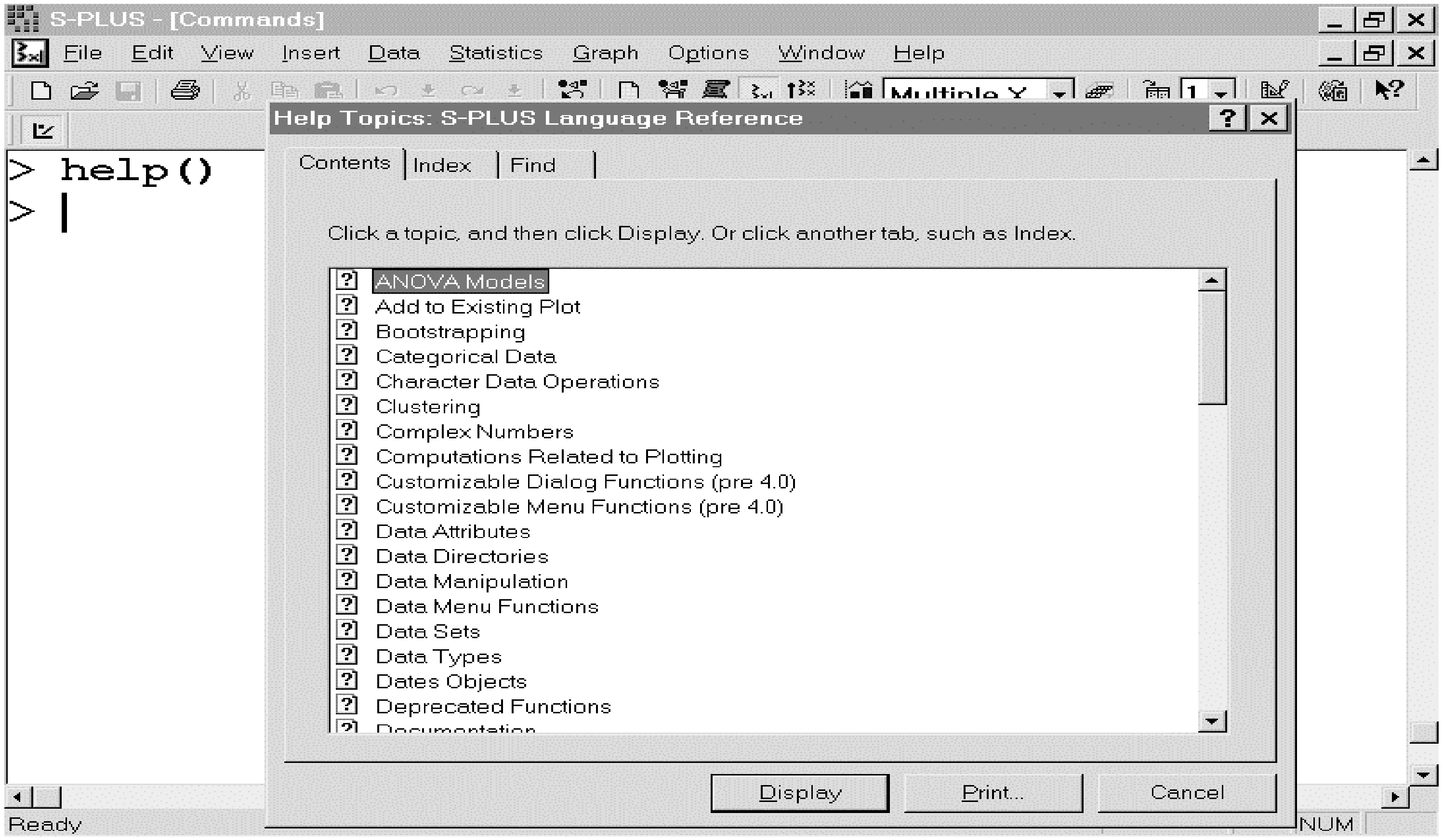
- There are several ways to quit S-PLUS
  - q()
  - Choose Exit from the File menu
  - Click on the Close box button.

Getting Help

- Go to the Help Menu on the Screen and select what you need.



- At the S-Plus prompt type **help().** It will starts the windows help application.



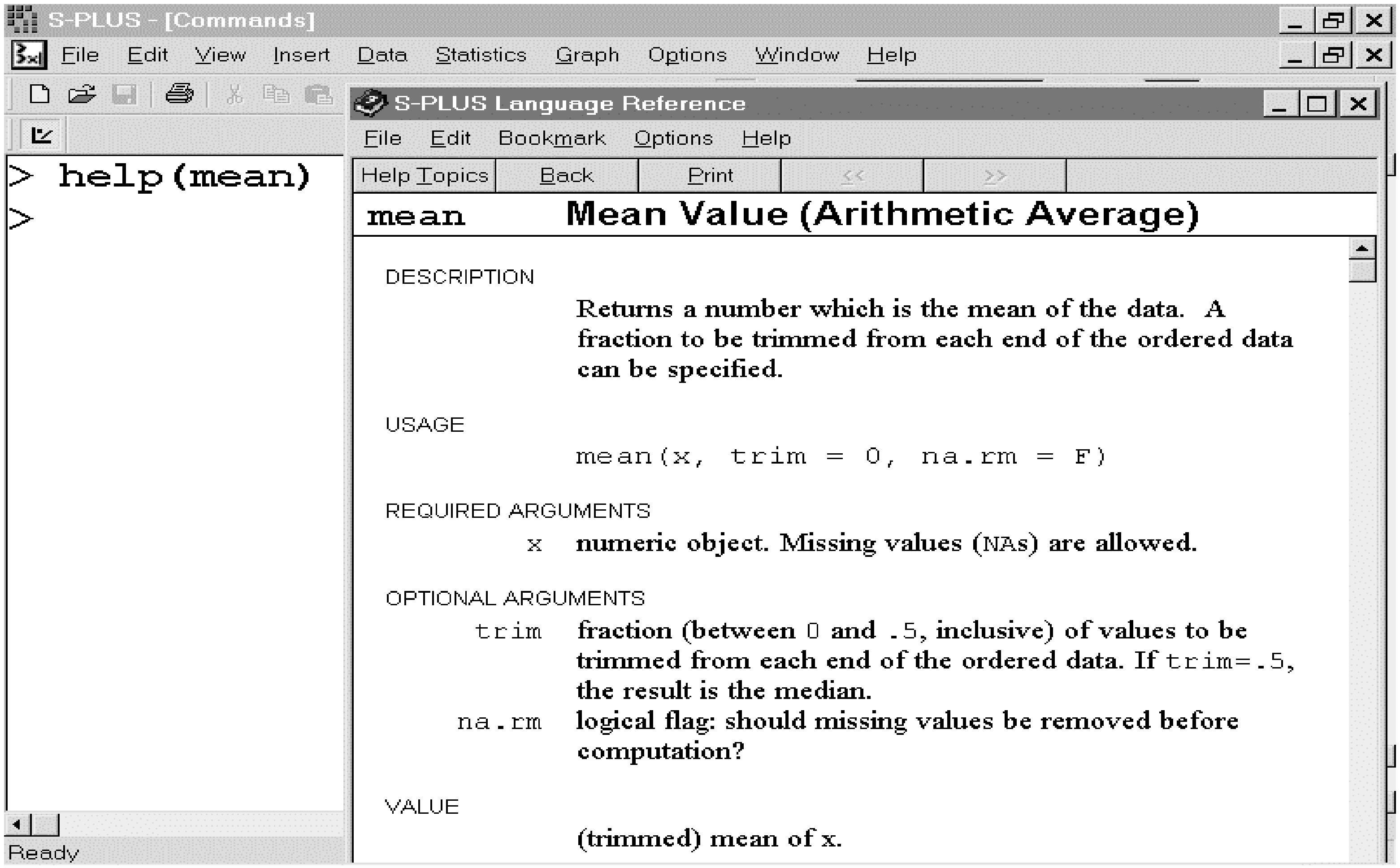
```
> help()
> |
```

Help Topics: S-PLUS Language Reference

Click a topic, and then click Display. Or click another tab, such as Index.

- ANOVA Models
- Add to Existing Plot
- Bootstrapping
- Categorical Data
- Character Data Operations
- Clustering
- Complex Numbers
- Computations Related to Plotting
- Customizable Dialog Functions (pre 4.0)
- Customizable Menu Functions (pre 4.0)
- Data Attributes
- Data Directories
- Data Manipulation
- Data Menu Functions
- Data Sets
- Data Types
- Dates Objects
- Deprecated Functions

- Also **help(something)** display the closest topic it can find.


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
S-PLUS Language Reference
File Edit Bookmark Options Help
Help Topics Back Print << >>
Ready

```
> help(mean)
>
```

mean Mean Value (Arithmetic Average)

DESCRIPTION

Returns a number which is the mean of the data. A fraction to be trimmed from each end of the ordered data can be specified.

USAGE

```
mean(x, trim = 0, na.rm = F)
```

REQUIRED ARGUMENTS

`x`	numeric object. Missing values (`NA`s) are allowed.

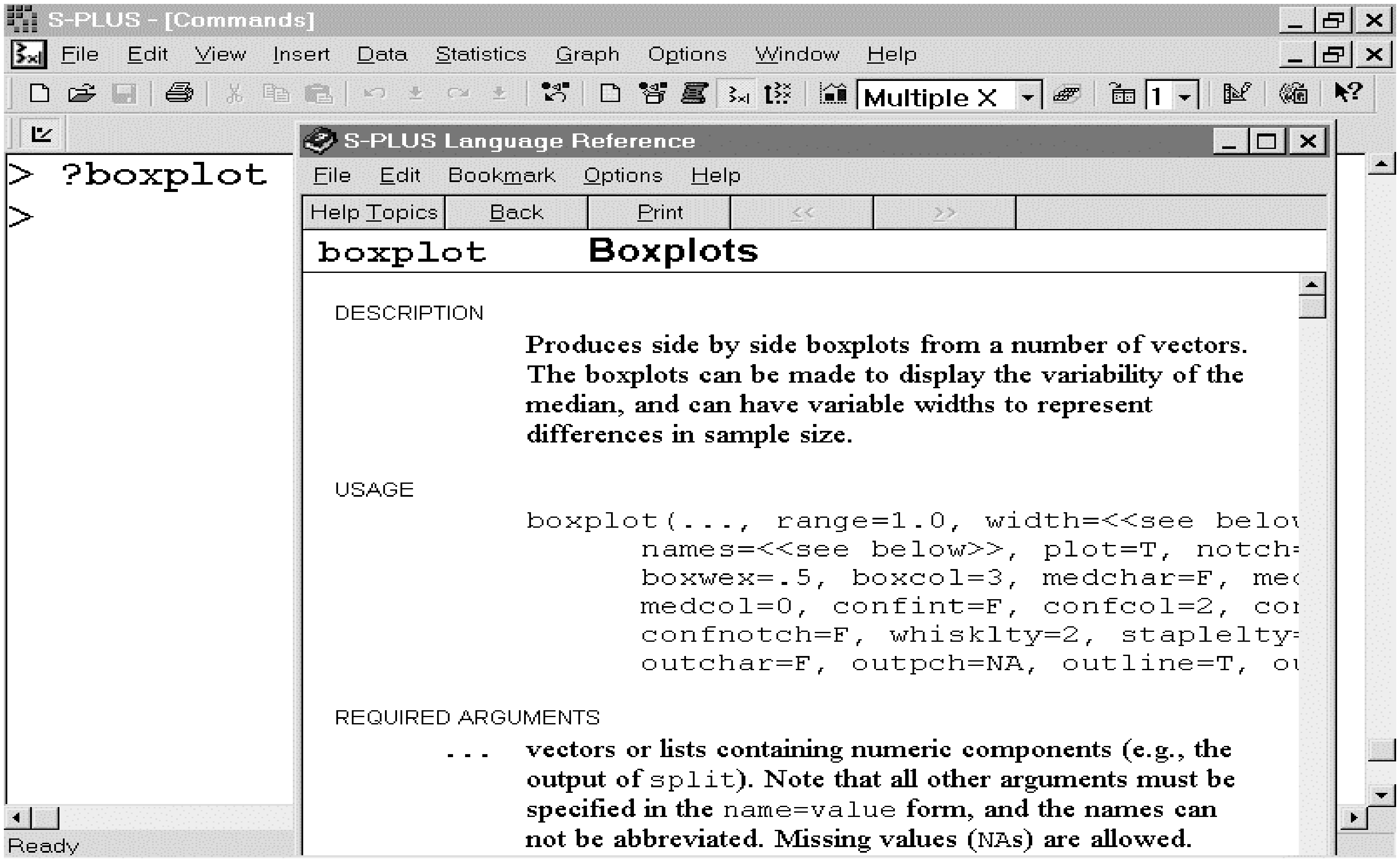
OPTIONAL ARGUMENTS

`trim`	fraction (between `0` and `.5`, inclusive) of values to be trimmed from each end of the ordered data. If `trim=.5`, the result is the median.
`na.rm`	logical flag: should missing values be removed before computation?

VALUE

(trimmed) mean of x.

- Also **?something** display the closest topic it can find.


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Multiple X
1
S-PLUS Language Reference
File Edit Bookmark Options Help
Help Topics Back Print << >>
Ready

```
> ?boxplot
>
```

boxplot Boxplots

DESCRIPTION

Produces side by side boxplots from a number of vectors. The boxplots can be made to display the variability of the median, and can have variable widths to represent differences in sample size.

USAGE

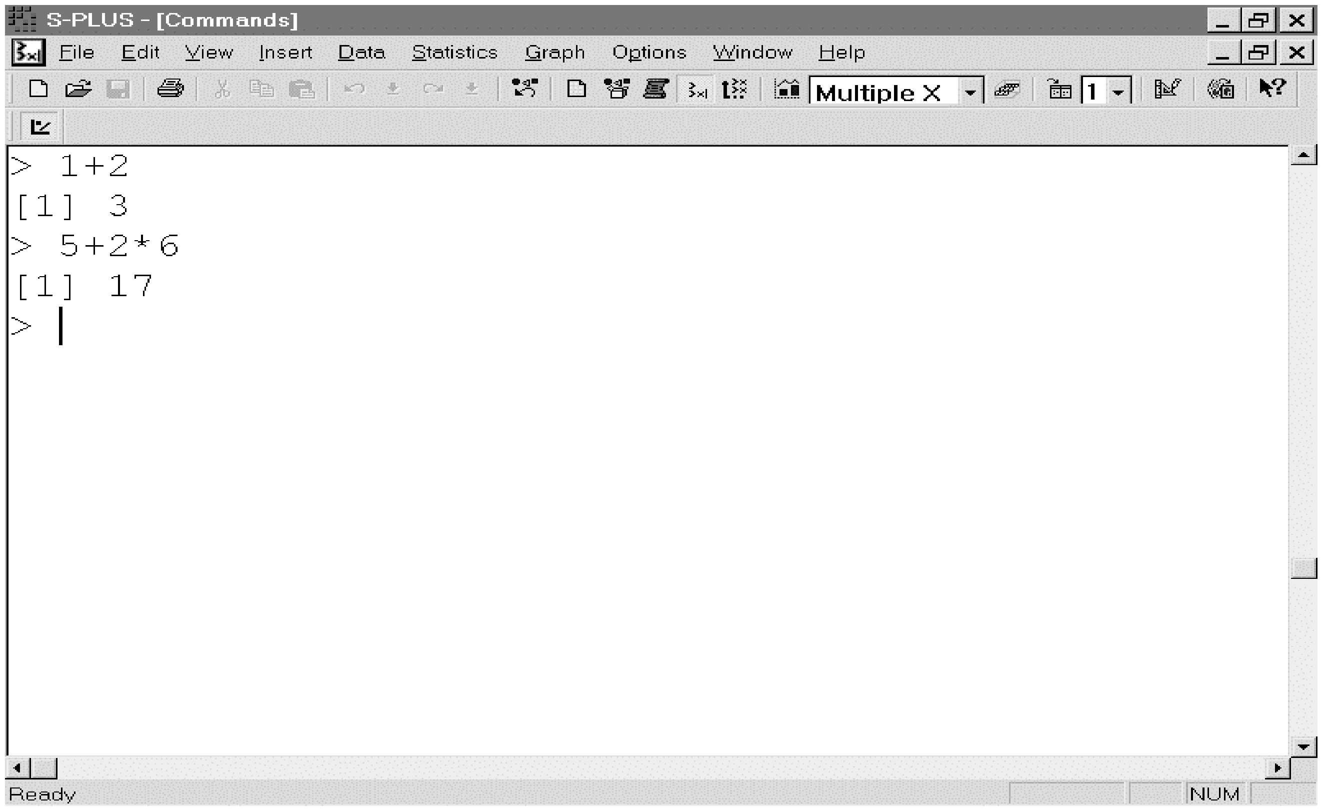
```
boxplot(..., range=1.0, width=<<see belo
      names=<<see below>>, plot=T, notch=
      boxwex=.5, boxcol=3, medchar=F, me
      medcol=0, confint=F, confcol=2, co
      confnotch=F, whisklty=2, staplelty=
      outchar=F, outpch=NA, outline=T, ou
```

REQUIRED ARGUMENTS

... vectors or lists containing numeric components (e.g., the output of `split`). Note that all other arguments must be specified in the `name=value` form, and the names can not be abbreviated. Missing values (`NA`s) are allowed.

Expressions and Objects in S-PLUS

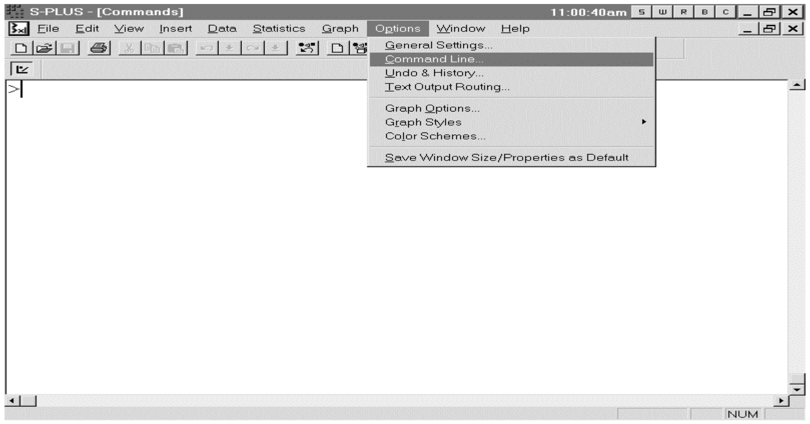
- Since S-PLUS is an interactive software program, at the prompt you can use S-Plus by typing expressions.


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Multiple X
1
Ready
NUM

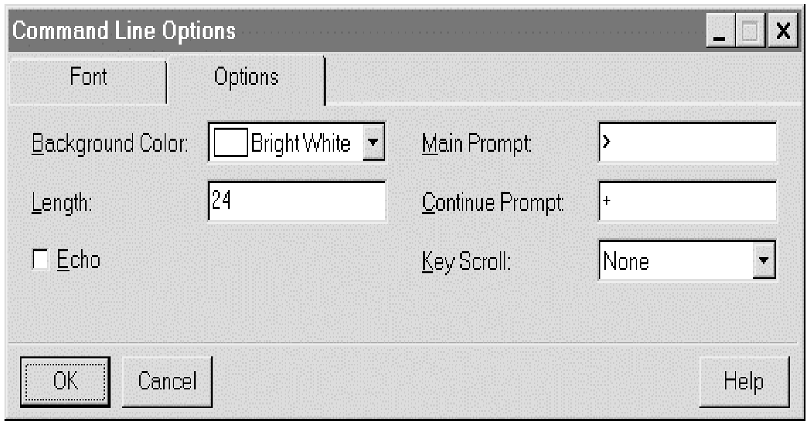
```
> 1+2
[1] 3
> 5+2*6
[1] 17
> 
```

- The default S-PLUS prompt is “>”
- The default continuation in S-PLUS is “+”
- To change the default you can do the following:
  - Go to Options
  - Choose Command Line
  - Click on the Options Tab
  - Change **Main Prompt** to “*****” and the **Continue Prompt** to “**&**”

Go to Options and Choose Command Line



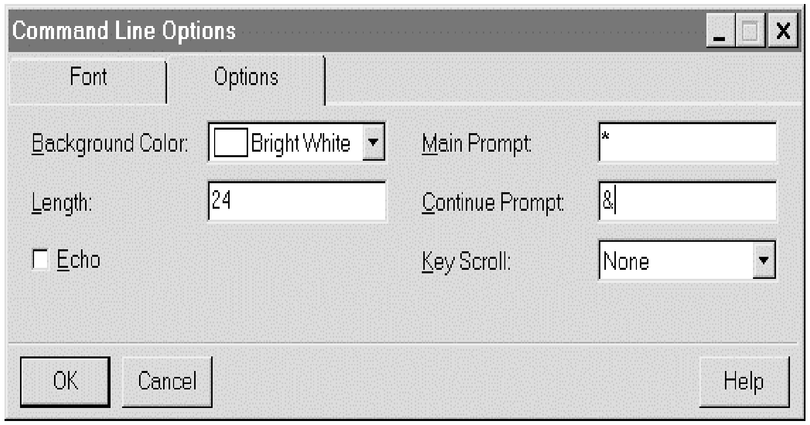
Click on the Options Tab



Background Color: Bright White
Main Prompt: >
Length: 24
Continue Prompt: +
☐ Echo
Key Scroll: None

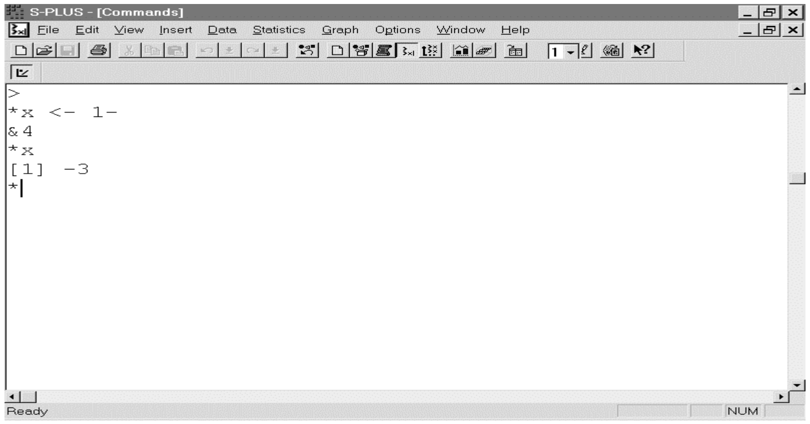
Change **Main Prompt** to “*****” and the

Continue Prompt to “**&**”



Background Color: Bright White
Main Prompt: *
Length: 24
Continue Prompt: &
☐ Echo
Key Scroll: None

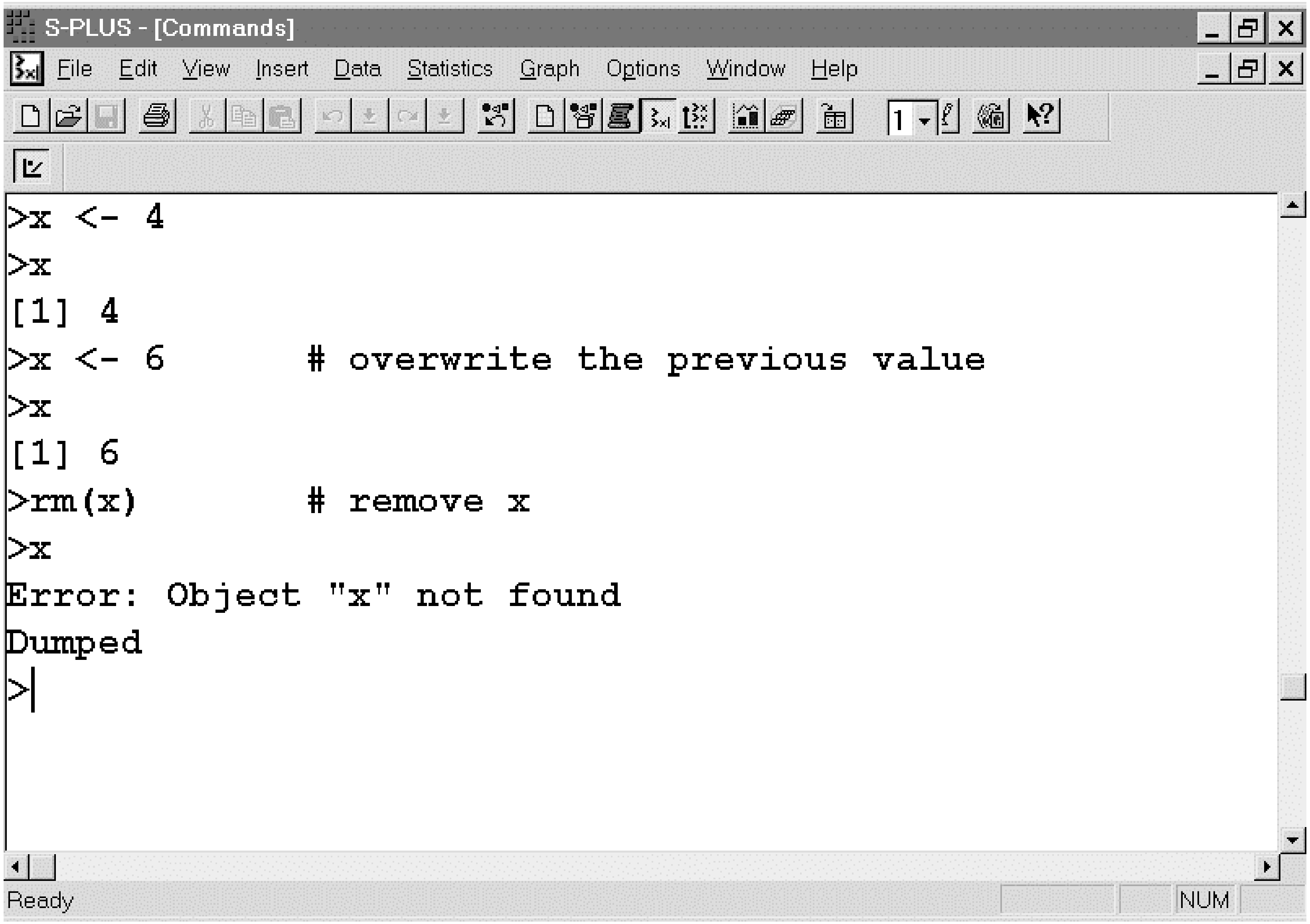
Result:


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Ready
NUM

```
>
*x <- 1-
&4
*x
[1] -3
*
```

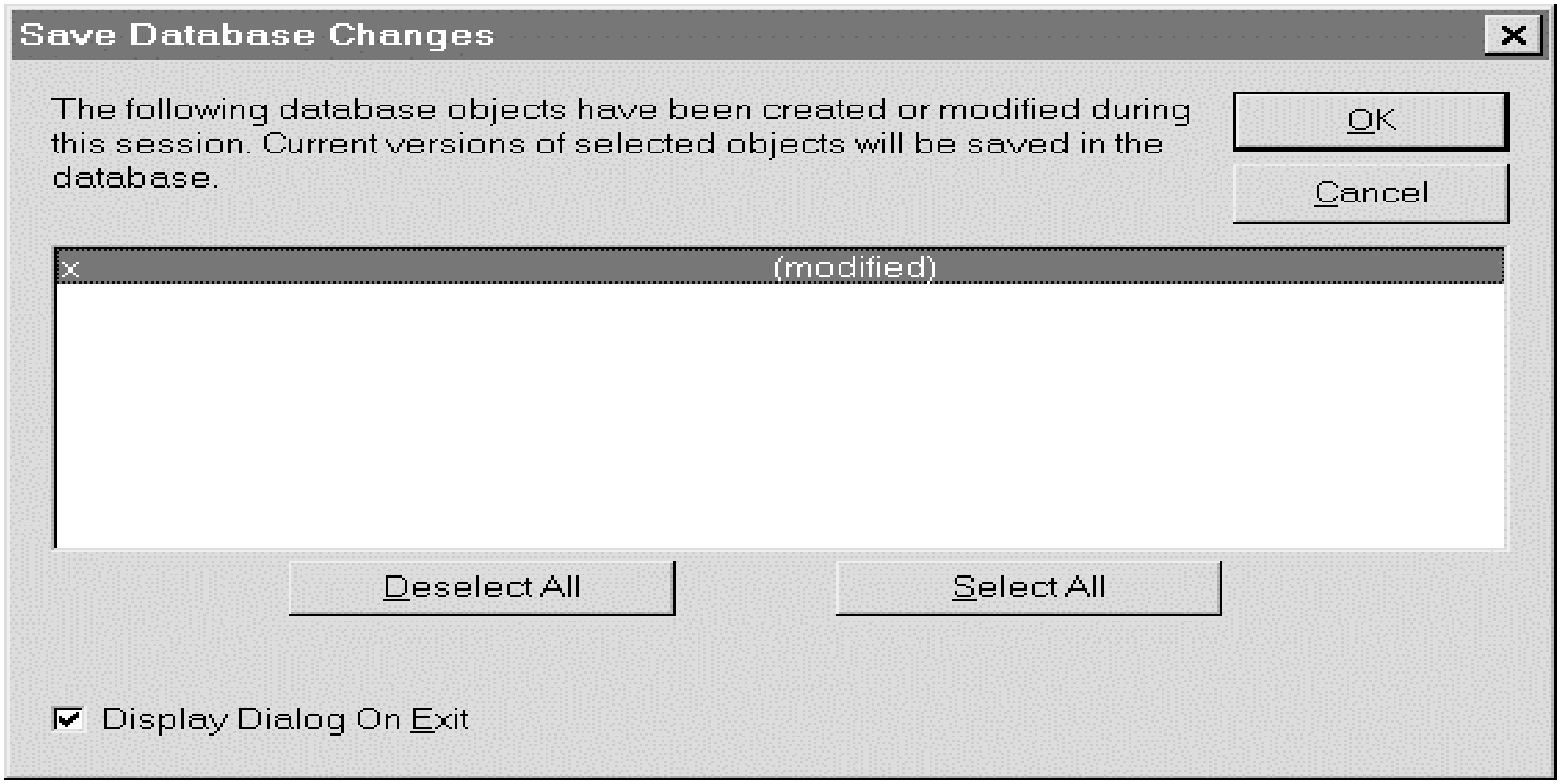
Storing and Removing Objects

- To store an object in S-Plus we have to give it a name.
- To assign names we use the assignment operator “<-” or underscore “_”.
- Assignments in S-Plus remains until removed or overwritten.
- To remove a variable you can use the **rm** function.


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Ready
NUM

```
>x <- 4
>x
[1] 4
>x <- 6        # overwrite the previous value
>x
[1] 6
>rm(x)         # remove x
>x
Error: Object "x" not found
Dumped
>
```

- If an existing object was changed, you have the option of making these changes permanent before you exit, or keeping the original value.



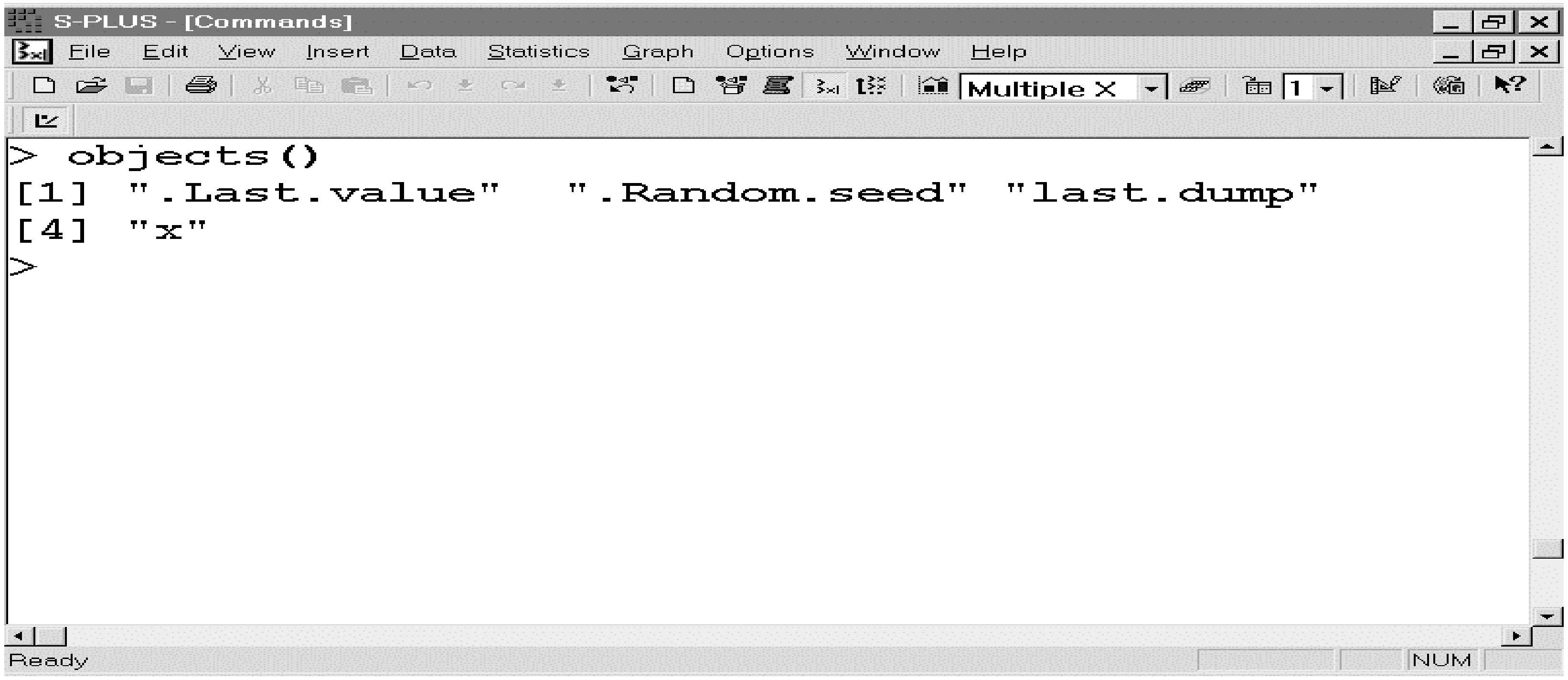
Save Database Changes

The following database objects have been created or modified during this session. Current versions of selected objects will be saved in the database.

x (modified)

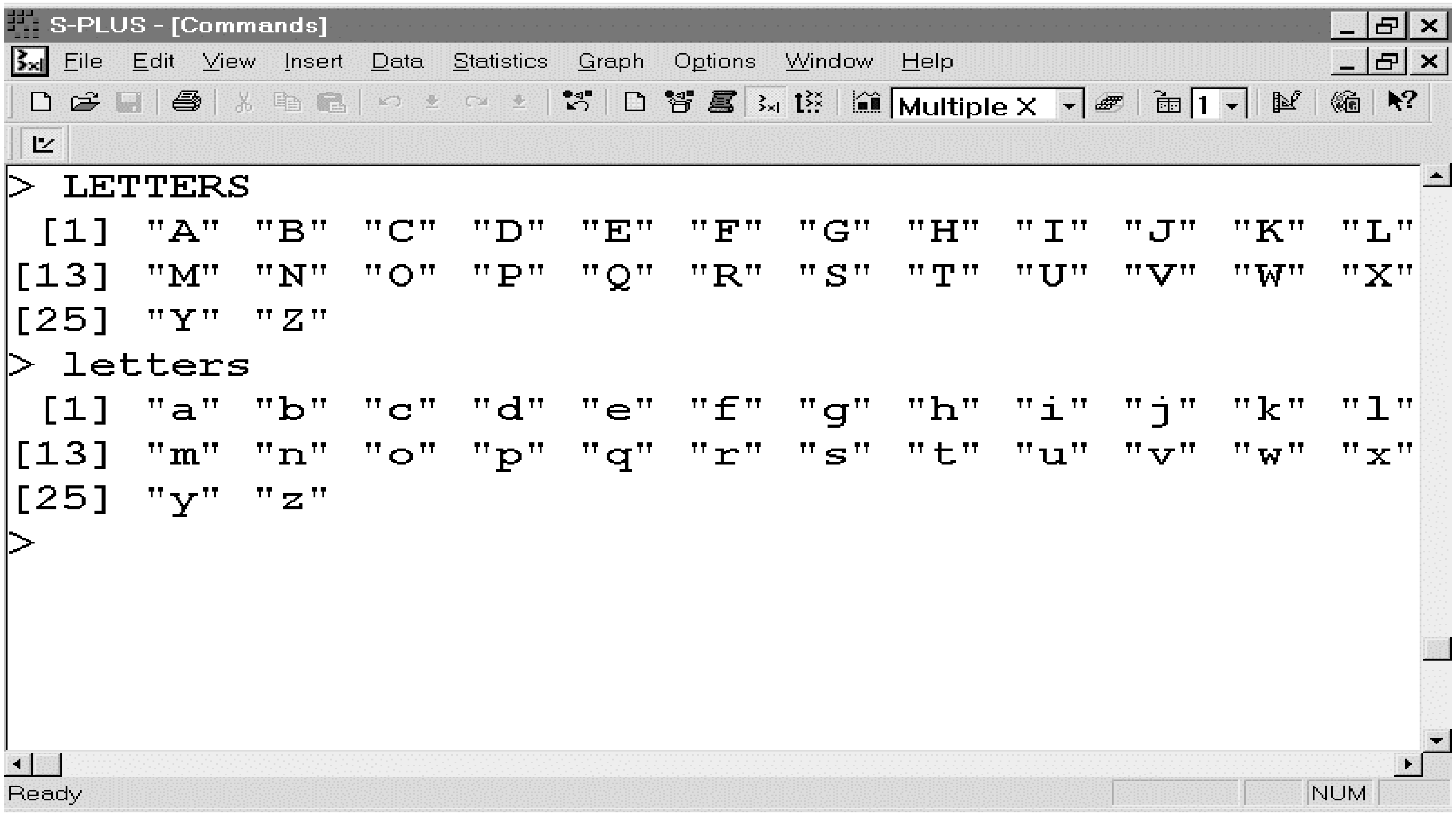
Finding Objects

- To see the names of the objects you have created in S-Plus you can use the function **objects**


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Multiple X
1
Ready
NUM

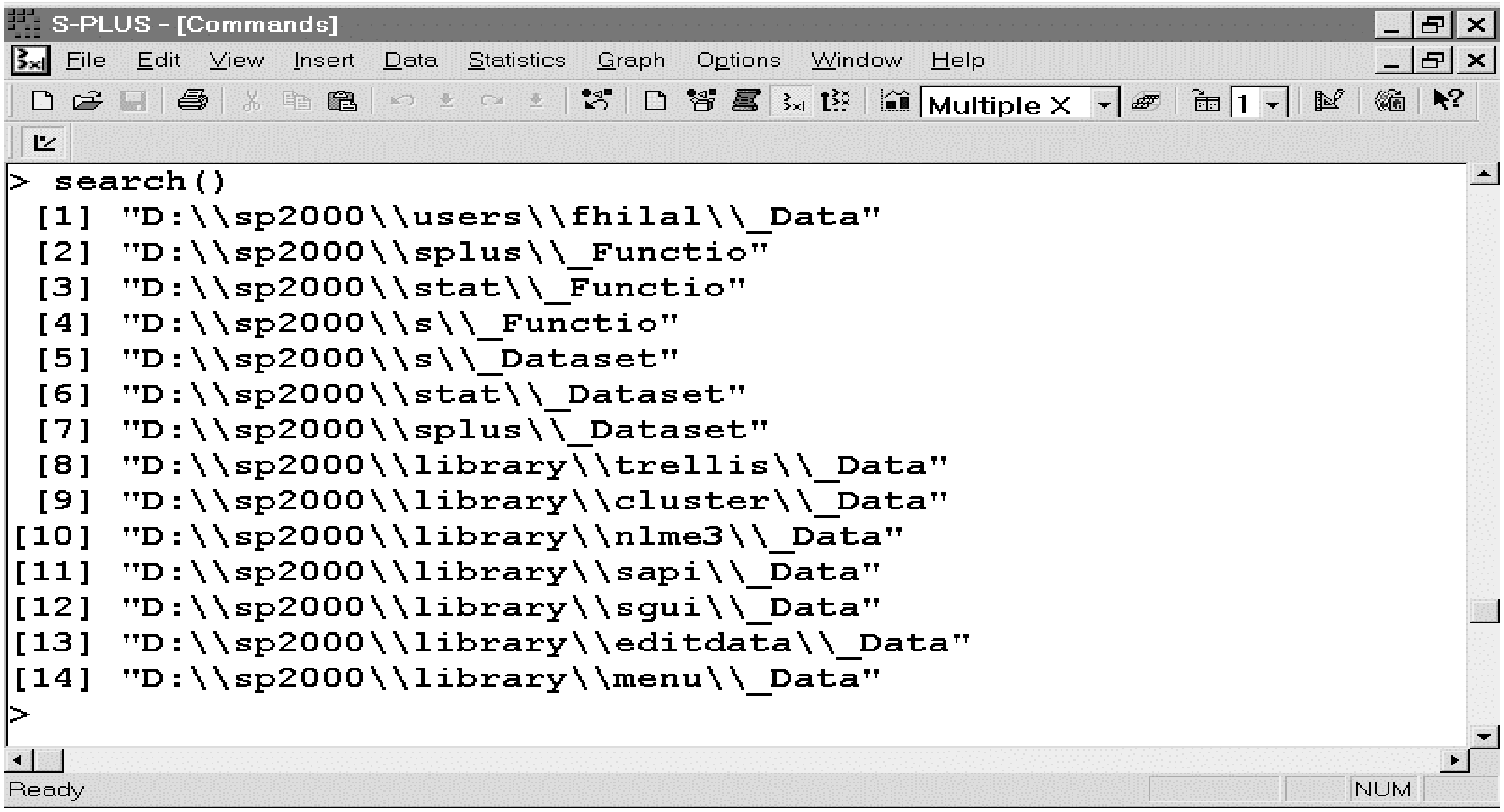
```
> objects()
[1] ".Last.value"   ".Random.seed" "last.dump"
[4] "x"
>
```

- S-Plus comes in with built-in functions and data sets like **LETTERS** and **letters** the upper and the lower case letters of the alphabet.


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Multiple X
1
Ready
NUM

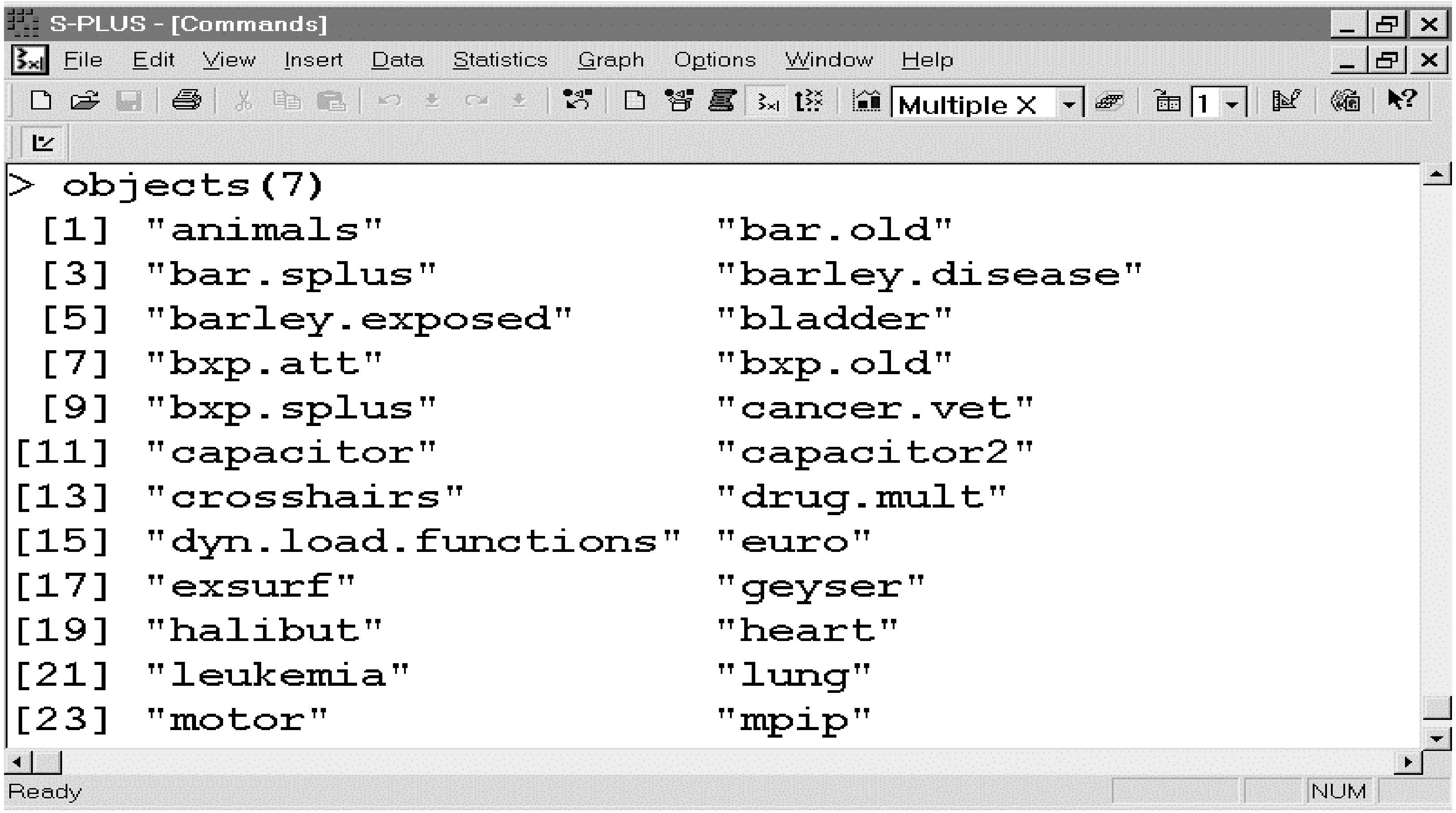
```
> LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L"
[13] "M" "N" "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X"
[25] "Y" "Z"
> letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
[13] "m" "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x"
[25] "y" "z"
>
```

- When you type an object name or function S-Plus looks through a sequence of directories called the search list. To see the current search list you can use the function **search()**


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Multiple X
1
Ready
NUM

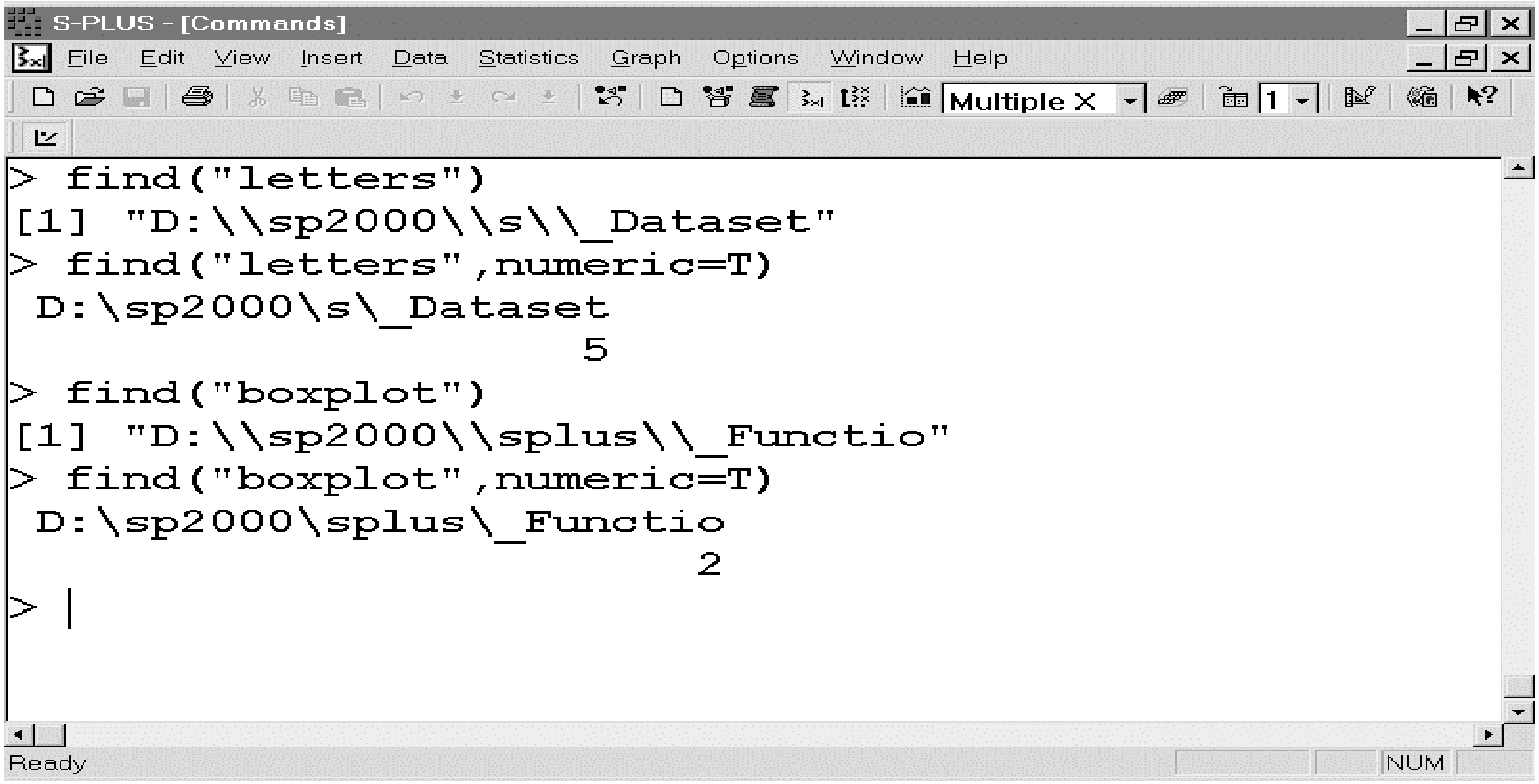
```
> search()
 [1] "D:\\sp2000\\users\\fhilal\\_Data"
 [2] "D:\\sp2000\\splus\\_Functio"
 [3] "D:\\sp2000\\stat\\_Functio"
 [4] "D:\\sp2000\\s\\_Functio"
 [5] "D:\\sp2000\\s\\_Dataset"
 [6] "D:\\sp2000\\stat\\_Dataset"
 [7] "D:\\sp2000\\splus\\_Dataset"
 [8] "D:\\sp2000\\library\\trellis\\_Data"
 [9] "D:\\sp2000\\library\\cluster\\_Data"
[10] "D:\\sp2000\\library\\nlme3\\_Data"
[11] "D:\\sp2000\\library\\sapi\\_Data"
[12] "D:\\sp2000\\library\\sgui\\_Data"
[13] "D:\\sp2000\\library\\editdata\\_Data"
[14] "D:\\sp2000\\library\\menu\\_Data"
>
```

- To list the names of the objects stored in the directories on the search list, use the **object** function with the directory number as an argument.


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Multiple X
1
Ready
NUM

```
> objects(7)
 [1] "animals"                "bar.old"
 [3] "bar.splus"              "barley.disease"
 [5] "barley.exposed"         "bladder"
 [7] "bxp.att"                "bxp.old"
 [9] "bxp.splus"              "cancer.vet"
[11] "capacitor"              "capacitor2"
[13] "crosshairs"             "drug.mult"
[15] "dyn.load.functions"     "euro"
[17] "exsurf"                 "geyser"
[19] "halibut"                "heart"
[21] "leukemia"               "lung"
[23] "motor"                  "mpip"
```

- To find a directory or directories that hold objects with a particular name we can use the **find** function.
- Also if you use the argument **numeric=T** the **find** function will give you the directory number.


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Multiple X
1
Ready
NUM

```
> find("letters")
[1] "D:\\sp2000\\s\\_Dataset"
> find("letters",numeric=T)
 D:\sp2000\s\_Dataset
                    5
> find("boxplot")
[1] "D:\\sp2000\\splus\\_Functio"
> find("boxplot",numeric=T)
 D:\sp2000\splus\_Functio
                        2
> |
```

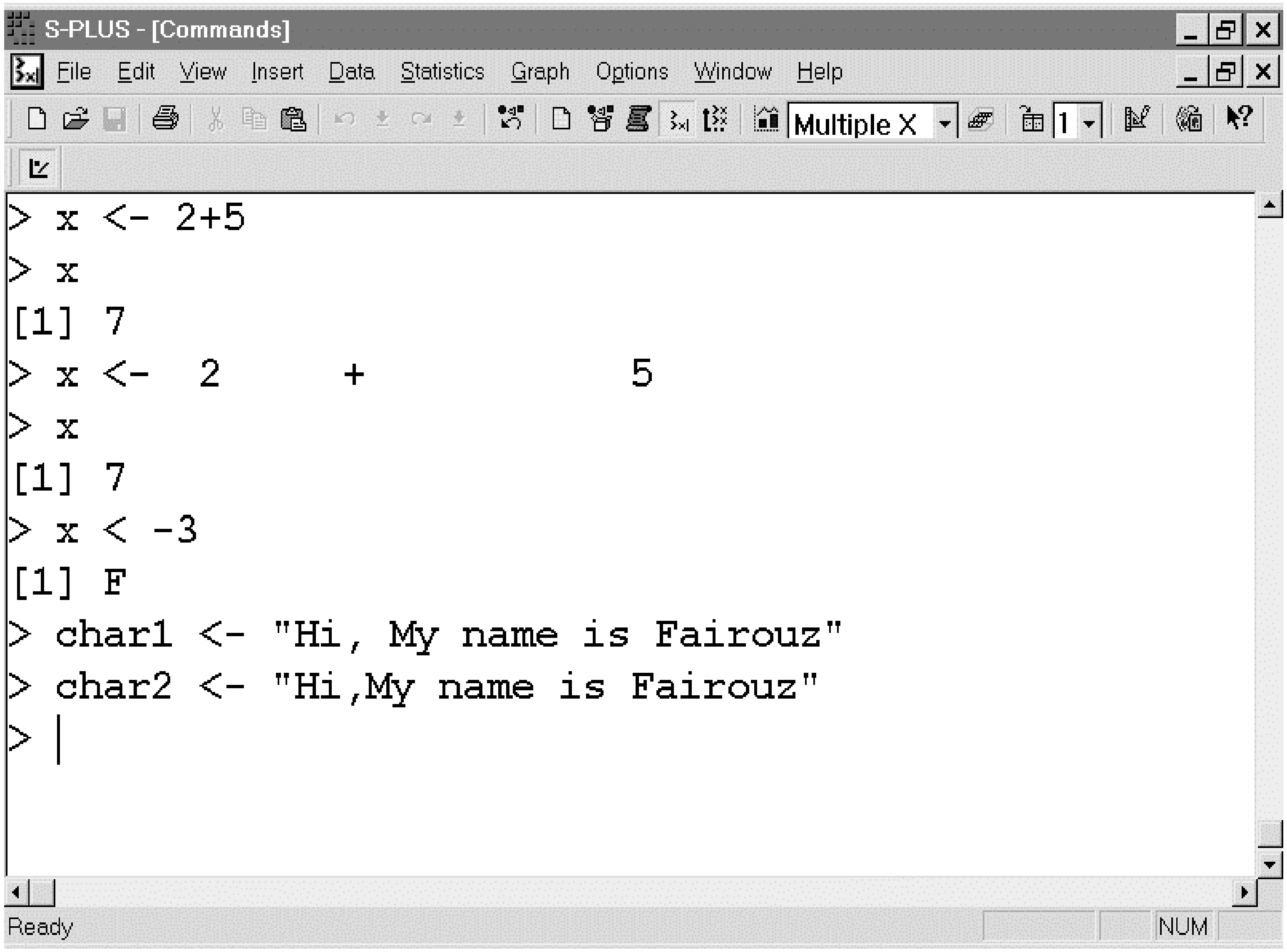
Legal Names IN S-Plus

- It can be any combination of letters, numbers and periods. But it **can't Start** with a number and no other symbols are allowed.
- There is no limit to the number of characters in an object name
- S-PLUS is case sensitive so **Age** is different than **age** and the two are different from **AGE**.
- You can't use the underscore "_" in the S-PLUS names.

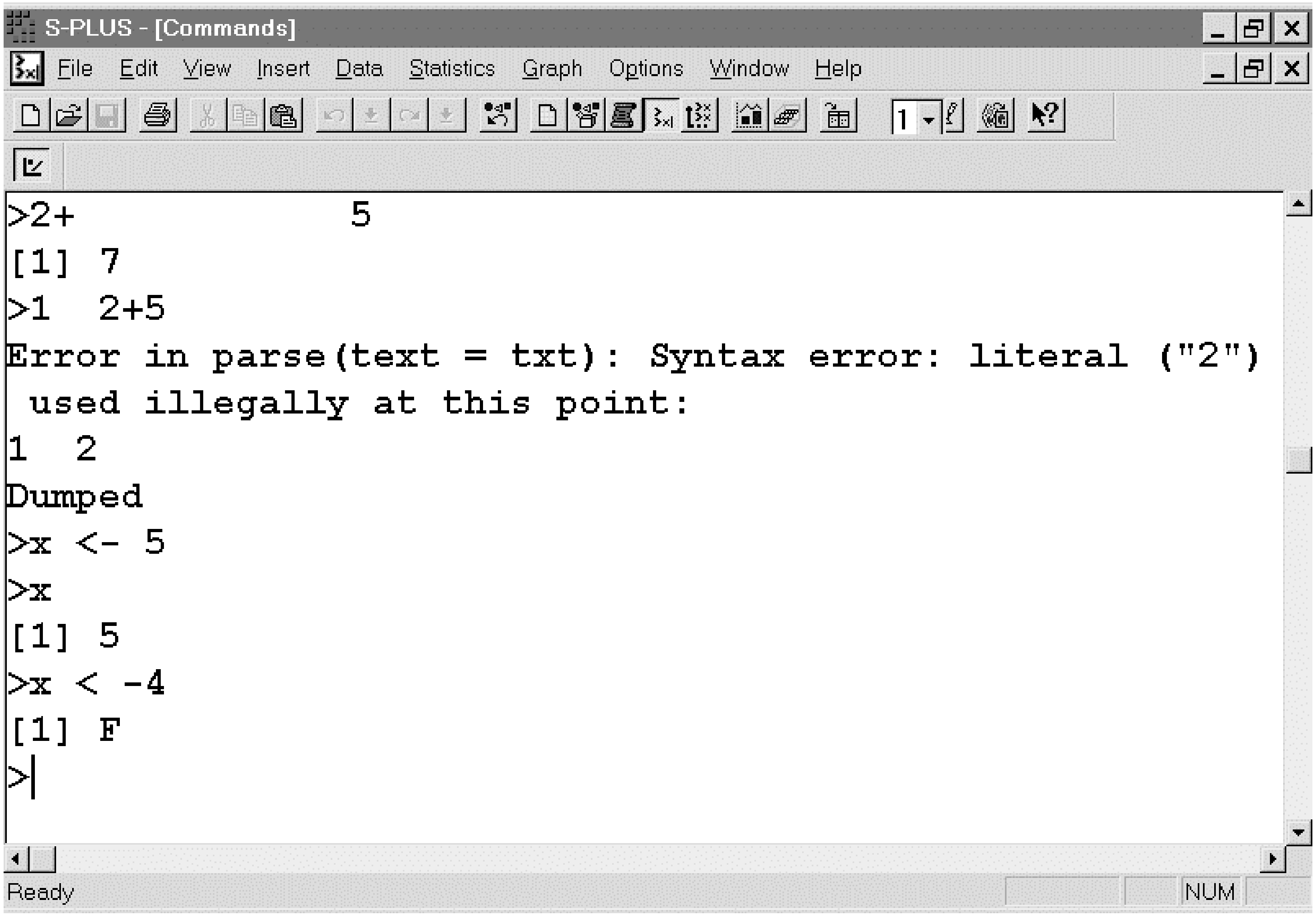
- Examples:
  - mydata
  - data.one
  - RandomNumbers
  - data.1.cit.gov
- DO NOT choose names that coincide with the built-in S-Plus functions like C,D, c, q, s, t (those are S-Plus functions that have a single -character name which users usually use them to name their own functions or data).

Basic Syntax and Conventions

- S-Plus ignores most spaces.
  - For example you can put an arbitrary amount of white spaces between an operator and a number.
- S-Plus does not allow a space in the middle of an object name or a number. It will interprets the resulting pieces as two names or two numbers. And S-Plus will give a Syntax error.
- You can not have a space in the two character assignment operator “<-” since “<” is the less than S-Plus operator and “-” is the S-Plus minus sign operator.
- Spaces within character string are recorded as you type them. So in the following example char1 and char2 are different.


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Multiple X
1
Ready
NUM

```
> x <- 2+5
> x
[1] 7
> x <-  2     +          5
> x
[1] 7
> x < -3
[1] F
> char1 <- "Hi, My name is Fairouz"
> char2 <- "Hi,My name is Fairouz"
> 
```


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Ready
NUM

```
>2+           5
[1] 7
>1  2+5
Error in parse(text = txt): Syntax error: literal ("2")
 used illegally at this point:
1  2
Dumped
>x <- 5
>x
[1] 5
>x < -4
[1] F
>
```

Note the error message.

Recalling Past Commands

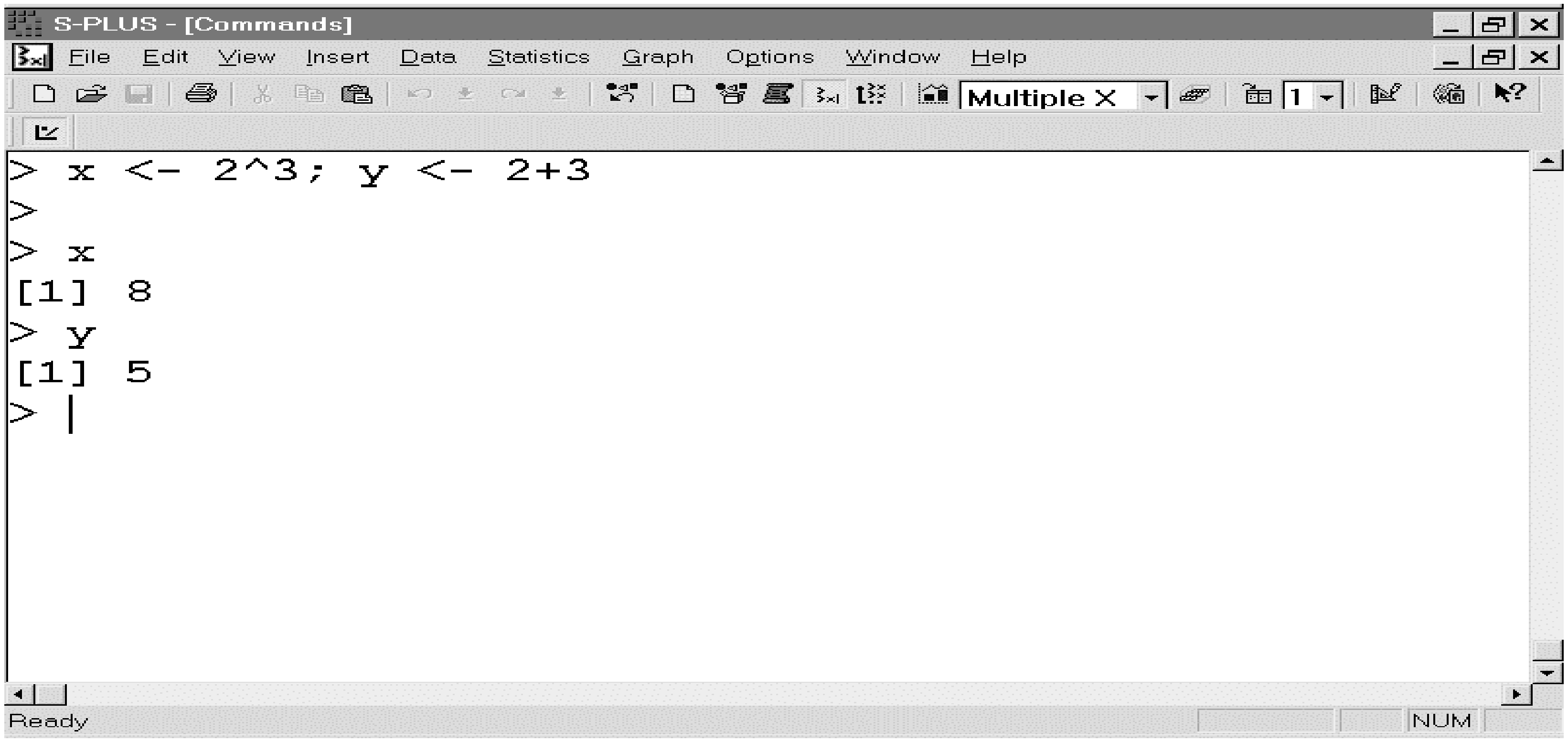
Action	**Keystroke**
Recall the previous line	Up Arrow
Go to the next line	Down Arrow
Recall the 10th line back	PageUp
Go forward 10 lines	PageDown
Search for selected text	F8

Editing Commands

Action	Keystroke
Move one character to the left	Left Arrow
Move one character to the right	Right Arrow
Erase right of cursor	Delete
Erase left of cursor	Backspace
Beginning of line	Home
End of line	End
Clear Command line and Search buffer	Esc

Multiple Commands On a Single Line

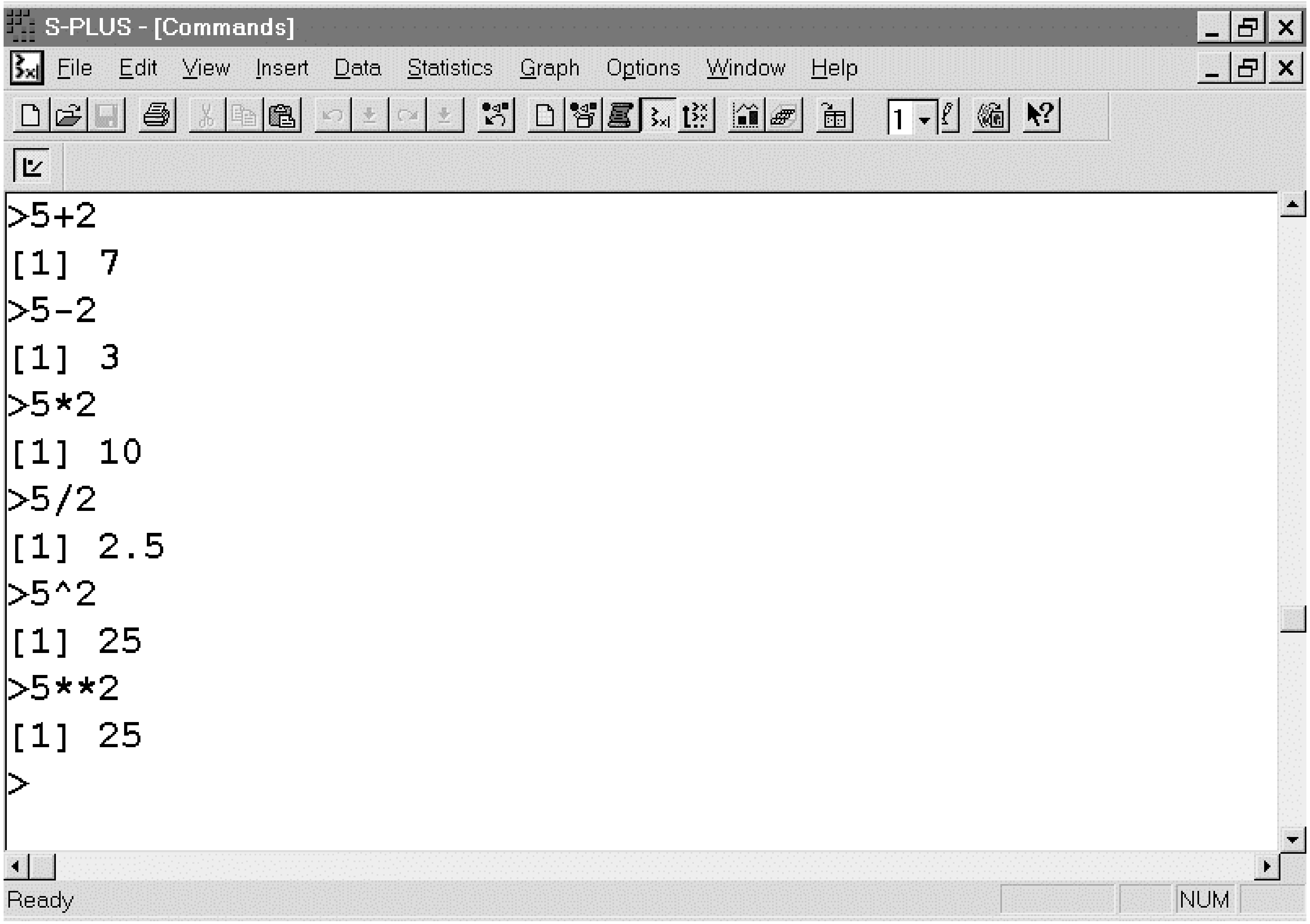
- Several commands can be typed on a single line by separating them by a **semicolon “;”**


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Multiple X
1
Ready
NUM

```
> x <- 2^3; y <- 2+3
>
> x
[1] 8
> y
[1] 5
>
```

Arithmetic Operators

- (+) is used for addition
- (-) is used for subtraction
- (*) is used for multiplication
- (/) is used for division
- (^) or (**) is used to raise values to a certain power


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Ready
NUM

```
>5+2
[1] 7
>5-2
[1] 3
>5*2
[1] 10
>5/2
[1] 2.5
>5^2
[1] 25
>5**2
[1] 25
>
```

Exercises: Introduction

Q1. Identify the directories in your current search list?

Q2. Find the directory that contains the function “glm”?

Q3. Get help on the function **objects** and use the *pattern* argument to identify all the objects in the above directory which contains “glm”

Data Objects

Contents:

- Types of data objects
- Modes, Attributes, and Classes

Types of data objects

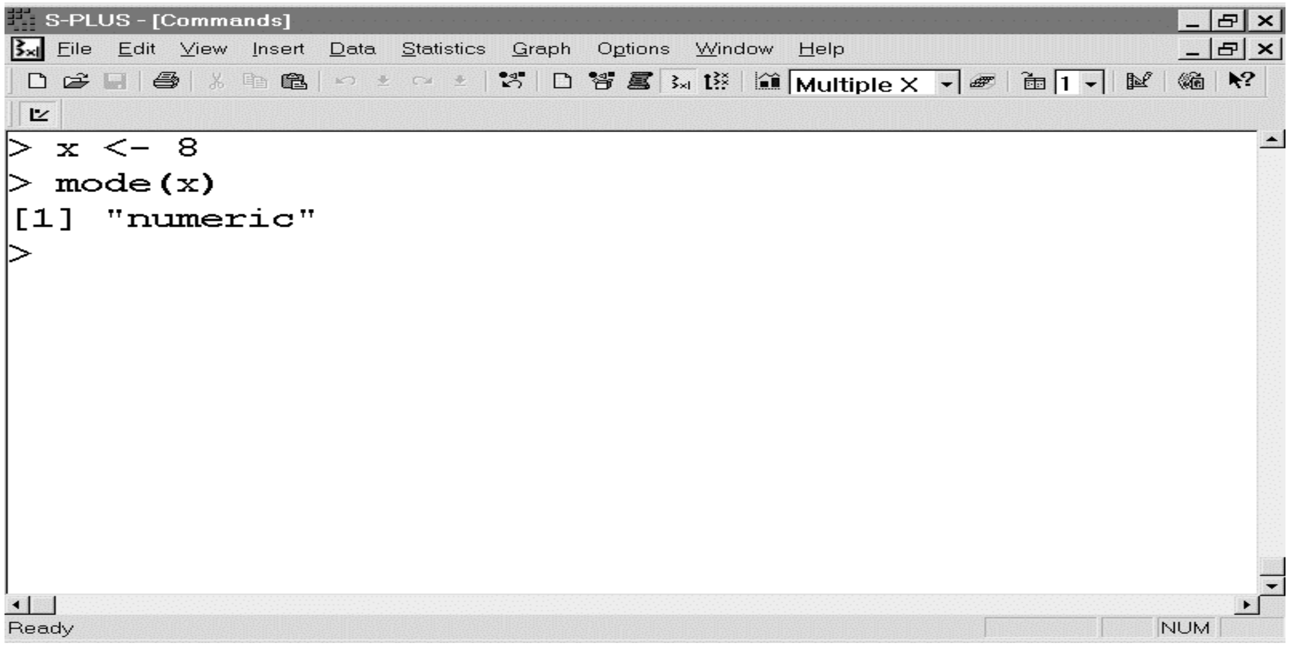
- There are seven basic types of data objects in S-Plus that can be classified into two categories
- Atomic objects that can contain values of only one kind.They are
  - Vector data objects
  - Matrix data objects
  - Array data objects
  - Factor data objects
  - Time Series data objects
- Non-Atomic objects that contain values of all kinds. They are
  - List data objects
  - Data Frame Objects

Modes, Attributes, and Classes

- **Mode:** The nature of the elements of the object, e.g. numeric, character
- **Attributes:** Characteristics and descriptive information about the object like the length of a vector, dimensions of a matrix etc.
- **Class:** The overall structure of the object, e.g. is it a matrix, a list or a linear model fit.

Modes

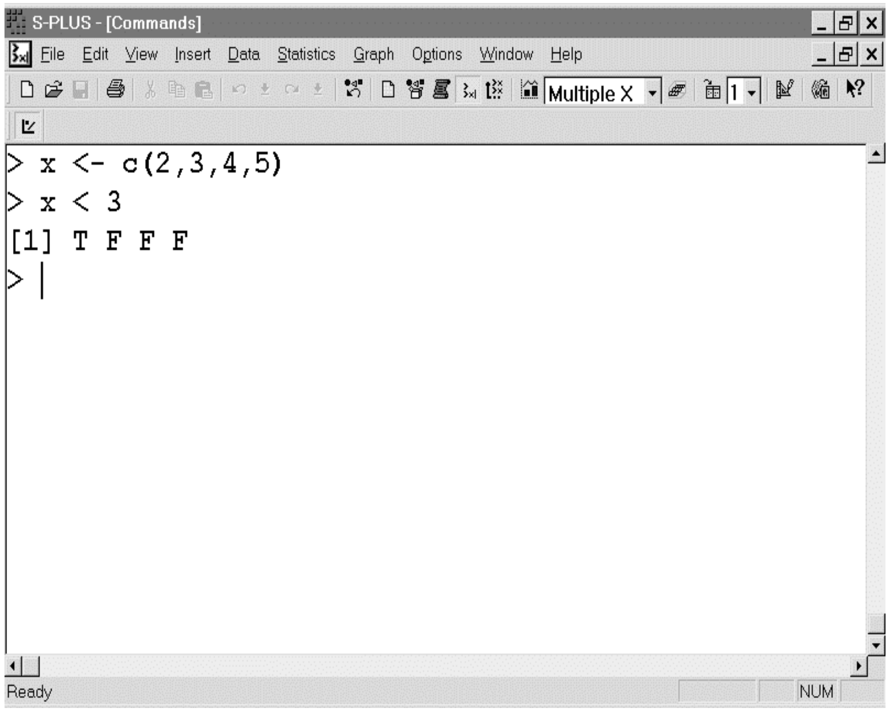
- The modes of values most commonly used in data analysis are as follows:
  - Logical, numeric, complex, character, NA and NULL
- The **mode** function returns the mode of the object


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Multiple X
1
Ready
NUM

```
> x <- 8
> mode(x)
[1] "numeric"
>
```

Logical Data Values

- Logical values, **TRUE (= T)** and **FALSE (= F)**, represent two-valued or binary data.
- Examples of types of data represented by logical values are
  - "yes" or "no"
  - "presence" or "absence"
  - "True" or "False"



S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Multiple X
Ready
NUM

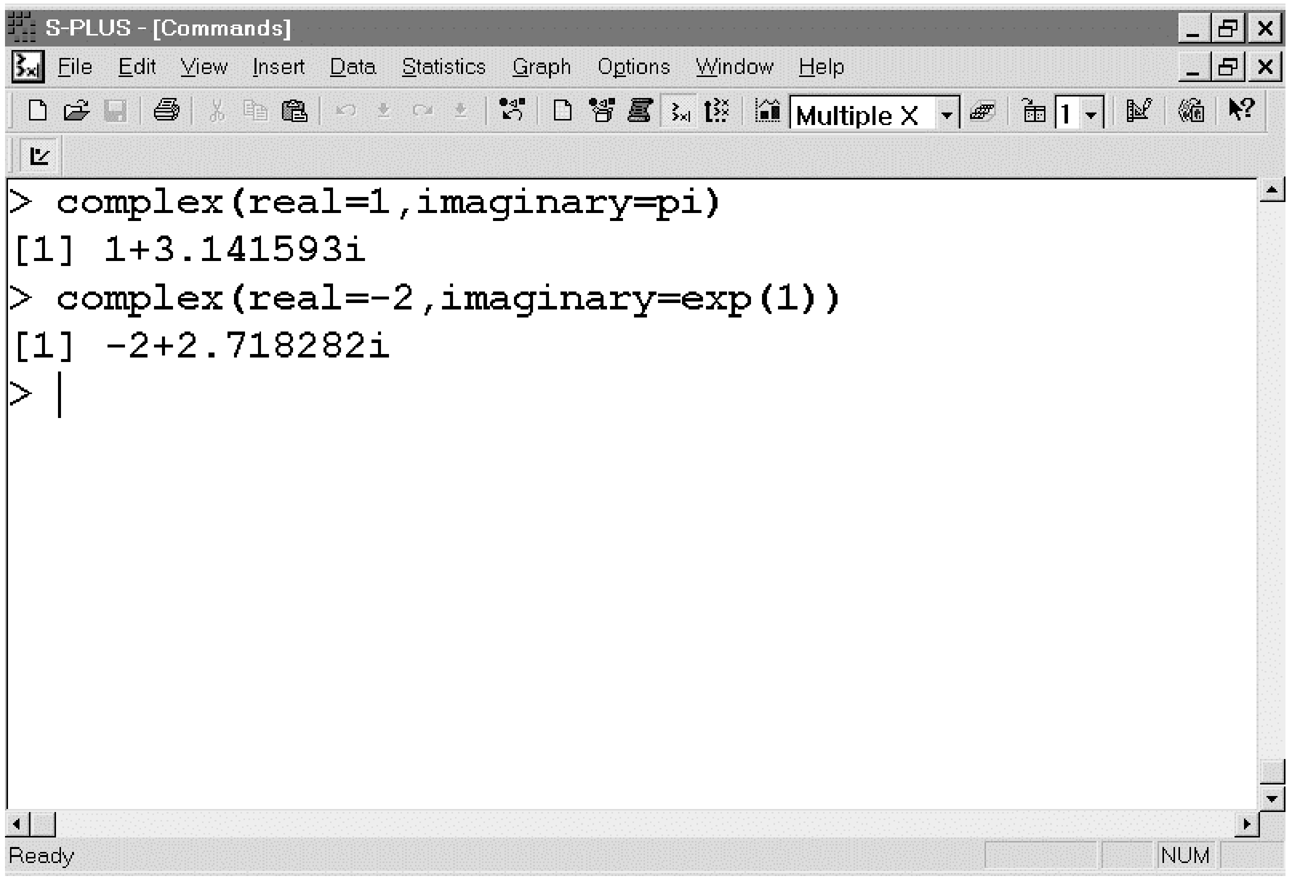
```
> x <- c(2,3,4,5)
> x < 3
[1] T F F F
> |
```

Numeric data Values

- They represent real numbers.
- They can be expressed in any of the following forms
  - Ordinary decimal numbers, such as 20, -4.5 or 15.678
  - As S-plus expression, such as pi, exp(1), or 1/3
  - Scientific Notation, which represents numbers as power of 10, such as 1e2 (=100) or 2e-2 (=0.02)
- Inf (infinity or ∞). This value can be either assigned to objects or it can be returned from the computation, such as 5/0

Complex data values

- They are specified in the form **a+bi** where **a** is the real part and **b** is the imaginary part.
- Examples:
  - 1+0i
  - 3-2i
  - 5.7+2.3i
- If we want to define a complex number with an imaginary part that is not expressed in decimal form we need to use the **complex** function as follows:


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Multiple X
1
Ready
NUM

```
> complex(real=1,imaginary=pi)
[1] 1+3.141593i
> complex(real=-2,imaginary=exp(1))
[1] -2+2.718282i
> |
```

Character data value

- Any character string enclosed in quotes (“ ”) or apostrophes (‘’) is a character value.
- You can have embedded spaces.

NA data value

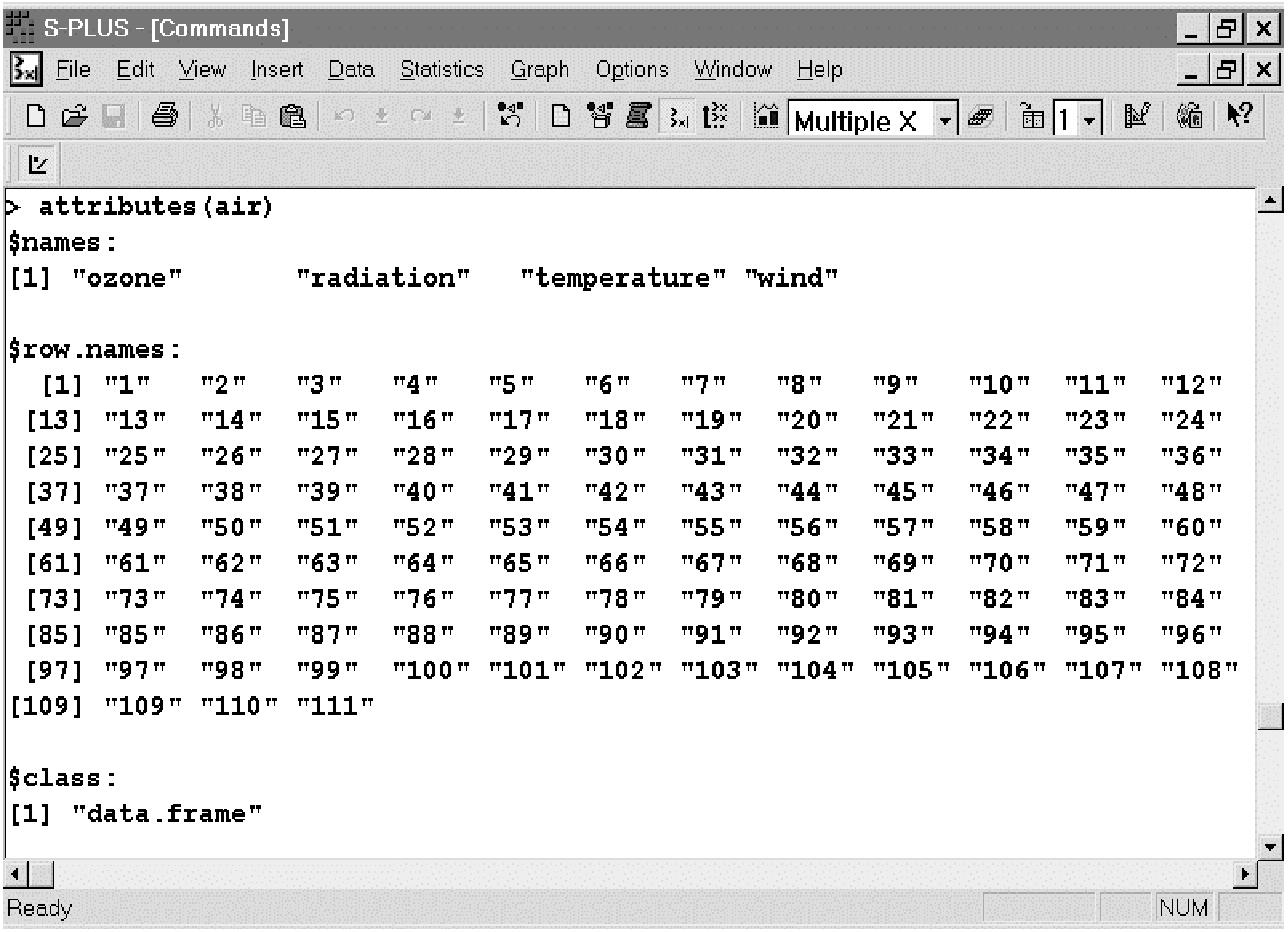
- It stands for “Not Available” and is the ***missing value*** code for logical, numerical, and complex data in S-Plus
- It can be entered directly as a value e.g. c(2,6,NA,7.5) or it can represent 0/0 that may results from intermediate computation.
- There is no missing value code for character value but we may represent it by “”.

NULL data value

- It represents a non-value.
- An example of this is the result from asking for the names associated with the values of a vector when there aren’t any.

Attributes

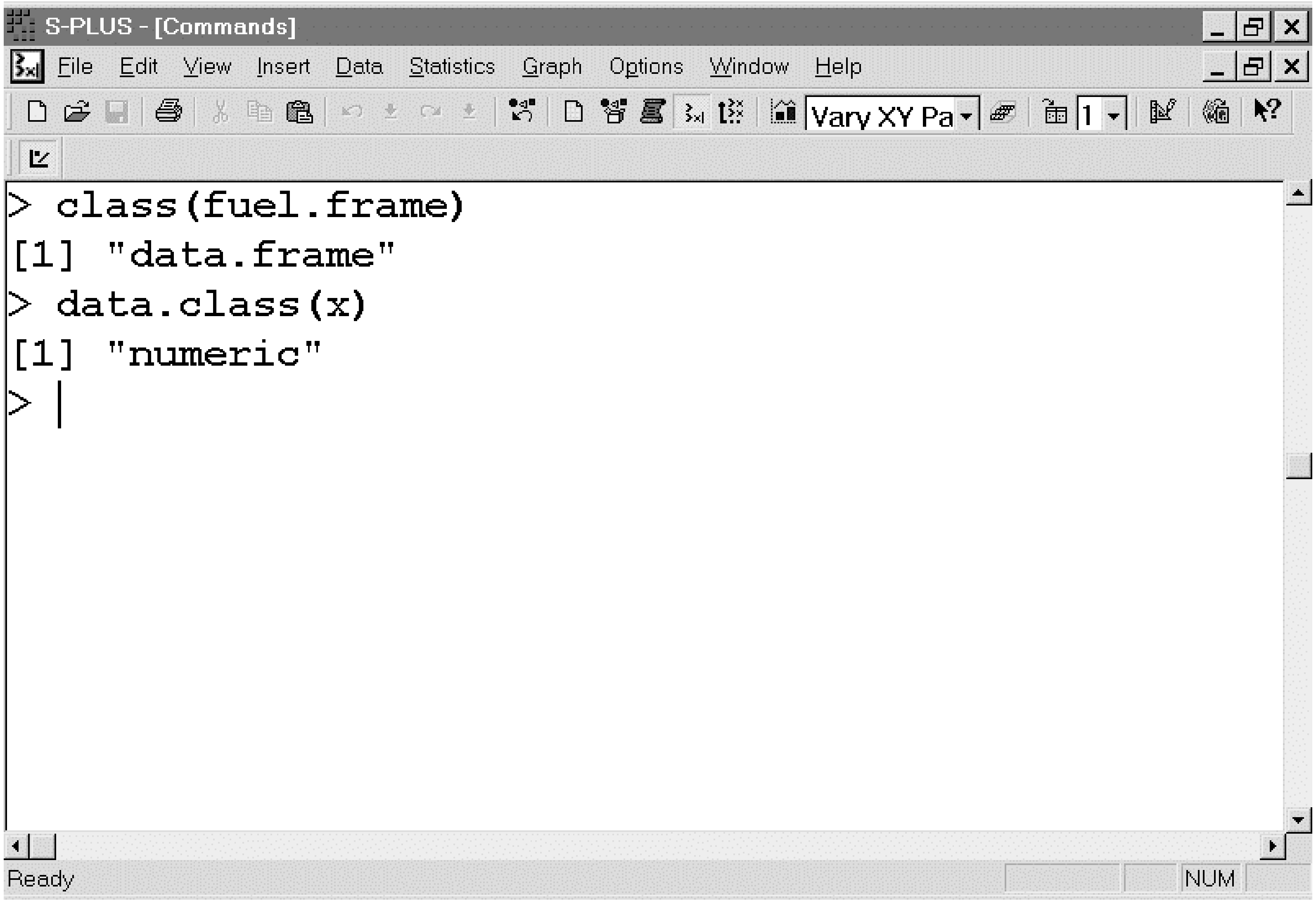
- It provides information on the object structure and contents
- It depends on the nature of the object
- The function **attributes** displays the attributes of an object


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Multiple X
1
Ready
NUM

```
> attributes(air)
$names:
[1] "ozone"       "radiation"   "temperature" "wind"

$row.names:
  [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "11"  "12"
 [13] "13"  "14"  "15"  "16"  "17"  "18"  "19"  "20"  "21"  "22"  "23"  "24"
 [25] "25"  "26"  "27"  "28"  "29"  "30"  "31"  "32"  "33"  "34"  "35"  "36"
 [37] "37"  "38"  "39"  "40"  "41"  "42"  "43"  "44"  "45"  "46"  "47"  "48"
 [49] "49"  "50"  "51"  "52"  "53"  "54"  "55"  "56"  "57"  "58"  "59"  "60"
 [61] "61"  "62"  "63"  "64"  "65"  "66"  "67"  "68"  "69"  "70"  "71"  "72"
 [73] "73"  "74"  "75"  "76"  "77"  "78"  "79"  "80"  "81"  "82"  "83"  "84"
 [85] "85"  "86"  "87"  "88"  "89"  "90"  "91"  "92"  "93"  "94"  "95"  "96"
 [97] "97"  "98"  "99"  "100" "101" "102" "103" "104" "105" "106" "107" "108"
[109] "109" "110" "111"

$class:
[1] "data.frame"
```

Class

- It describes the overall structure of the object
- The function **class** displays the class of sophisticated types of object that has an explicit type attribute.
- The function **data.class** will display the class of an object based on its attribute. This function is used with the objects that does not have an explicit class attribute like vectors, matrices, time series and factors.


S-PLUS - [Commands]
File Edit View Insert Data Statistics Graph Options Window Help
Vary XY Pa
1
Ready
NUM

```
> class(fuel.frame)
[1] "data.frame"
> data.class(x)
[1] "numeric"
> 
```

Atomic Data Objects

Vectors data Objects

- It is a set of numbers, character values, logical values, etc..
- Vectors must be of a *single mode*.
- Vectors have three attributes: **length, mode and name**
  - Length: gives the number of values in the vector and can be displayed by the **length** function
  - Mode: gives the kind of values and can be displayed with the **mode** function.
  - Names: gives the value labels and can be displayed by the **names** function.

Creating Vectors

- The following table summarizes the most useful functions for creating vectors

Function	Description
scan	read values any mode
c	Combine values any mode
rep	Repeat values any mode
seq	Numeric
vector	Initialize vectors
logical	Initialize logical vectors
numeric	Initialize numeric vectors
complex	Initialize complex vectors
character	Initialize "character" vectors

Examples: Scan Function

```
> x <- scan()
1: 2 4 6 8 10
6:
> x
[1]  2  4  6  8 10
> x <- scan(what=character())
1: a b c
4:
> x
[1] "a" "b" "c"
```

Examples: c Function

```
> x <- c(1,3,5)
> x
[1] 1 3 5
> x <- c(T,F,T,T)
> x
[1] T F T T
> x <- c("a","b","c")
> x
[1] "a" "b" "c"
```

Examples: rep Function

```
> x <- rep(NA,3)
> x
[1] NA NA NA
> x <- rep(2,6)
> x
[1] 2 2 2 2 2 2
> x <- rep(c(1,2,3),2)
> x
[1] 1 2 3 1 2 3
> x <- rep(c("y","n"),c(2,4))
> x
[1] "y" "y" "n" "n" "n" "n"
```

Examples: seq Function

```
> x <- seq(-1,1,0.4)
> x
[1] -1.0 -0.6 -0.2  0.2  0.6  1.0
> x <- seq(-2,2,length=5)
> x
[1] -2 -1  0  1  2
> x <- seq(0,by=0.1,length=4)
> x
[1] 0.0 0.1 0.2 0.3
```

```
> x <- seq(1,by=0.02,along=1:6)
> x
[1] 1.00 1.02 1.04 1.06 1.08 1.10
> x <- 1:3
> x
[1] 1 2 3
> x <- 3:-4
> x
[1]  3  2  1  0 -1 -2 -3 -4
> x <- 3.2:5
> x
[1] 3.2 4.2
```

Examples: vector Function

```
> x <- vector("numeric",5)
> x
[1] 0 0 0 0 0
> x <- vector("character",5)
> x
[1] "" "" "" "" ""
> x <- vector("complex",3)
> x
[1] 0+0i 0+0i 0+0i
> x <- vector("logical",3)
> x
[1] F F F
```

Examples: logical, numeric, complex and character Functions

```
> x <- numeric(3)
> x
[1] 0 0 0
> x <- character(3)
> x
[1] "" "" ""
> x <- logical(3)
> x
[1] F F F
> x <- complex(3)
```

Examples: vector attributes

```
> x <- 1:8
> x
[1] 1 2 3 4 5 6 7 8
> mode(x)
[1] "numeric"
> length(x)
[1] 8
> names(x) <- letters[1:8]
> x
 a b c d e f g h
 1 2 3 4 5 6 7 8
```

Subscripting Vectors

- To subscribe a vector we do
  - **Vec[subscript]**
- Ways of subscribing a vector
  - **Blank**: All of the elements of the vector are selected
  - **Positive Integers**: Select the elements in the vector specified by the subscript
  - **Negative Integers**: Select the elements in the vector that are not specified by the subscript
  - **Logical Values**: Select all elements of the vector corresponding to TRUE values in the subscript
  - **Character Strings**: Select the elements of the vector specified by the subscript, based on the names attribute of the vector.

Example: Subscript

```
> x[]     # Blank
 a b c d e f g h
 1 2 3 4 5 6 7 8
> x[c(8,3,2)]   # Positive Integers
 h c b
 8 3 2
> x[3:5]   # Positive Integers
 c d e
 3 4 5
> x[-c(1,7)]  # Negative Integers
 b c d e f h
 2 3 4 5 6 8
```

```
> x[c(T,T,F,T,F,T,F,F)]  # Logical Values
 a b d f
 1 2 4 6
> x[x>5]  # Logical Values
 f g h
 6 7 8
> x[c("b","c")]
 b c
 2 3
```

Matrix Data Objects

- It is a two-way array. They are used to arrange values by rows and columns in a rectangular table.
- Matrices have four attributes:length, mode,dim and dimnames
  - Length: gives the number of values in the matrix and can be displayed by the **length** function
  - Mode: gives the kind of values and can be displayed with the **mode** function.
  - Dim: gives the number of rows and columns and can be displayed with the **dim** function.
  - Dimnames: gives the row and column name and can be displayed by the **dimnames** function.

Creating Matrices

- Using the “**matrix**” function
- Using “**dim**” function
- Using “**rbind**” and the “cbind” function

Creating Matrices: using “matrix” function

- matrix(data , nrow=n ,ncol=m, byrow=F, dimnames=NULL)
- As default the values are placed in the matrix column by column. That is all rows of the first column are filled, then the rows of the second column are filled an so on.
- To fill the matrix row by row, set byrow to equal T.

Example: matrix Function

```
> mat <- matrix(1:12,nrow=2)
> mat
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    5    7    9   11
[2,]    2    4    6    8   10   12
> mat <- matrix(1:12,nrow=2,byrow=T)
> mat
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    4    5    6
[2,]    7    8    9   10   11   12
```

Creating Matrices: using rbind & cbind functions

- Combine vectors of matrices into a single matrix using
  - **rbind** function: combine vectors raw by raw
  - **cbind** function: combine vectors column by column
- Note: when vectors of different lengths are combined using cbind or rbind, the shorter ones gets replicated so the matrix is “filled in”.

Example: cbind Function

```
> x <- c(2,3,4,5)
> y <- c(8,10,5,4)
> mat <- rbind(x,y)
> mat
  [,1] [,2] [,3] [,4]
x    2    3    4    5
y    8   10    5    4
```

Example: rbind Function

```
> x <- c(2,3,4,5)
> y <- c(8,10,5,4)
> mat <- cbind(x,y)
> mat
     x  y
[1,] 2  8
[2,] 3 10
[3,] 4  5
[4,] 5  4
```

Creating Matrices: with “dim” function

- Create a matrix from an existing vector we can use the **dim** function to set the “dim“ attribute.

Example: dim Function

```
> mat <- c("a","b","c","d","e","f")
> dim(mat) <- c(2,3)
> mat
     [,1] [,2] [,3]
[1,] "a"  "c"  "e"
[2,] "b"  "d"  "f"
```

Example: matrix attributes

```
> mat <- matrix(seq(-2,2,length=6),nrow=2)
> mat
     [,1] [,2] [,3]
[1,] -2.0 -0.4  1.2
[2,] -1.2  0.4  2.0
> dim(mat)
[1] 2 3
> mode(mat)
[1] "numeric"
```

```
> length(mat)
[1] 6
> row.names <- paste("row",letters[1:2])
> col.names <- paste("col",letters[1:3])
> dimnames(mat) <- list(row.names,col.names)
> mat
      col a col b col c
row a  -2.0  -0.4   1.2
row b  -1.2   0.4   2.0
```

Subscripting Matrices

- Each dimension of a matrix may be subscripted in the same way as vectors. If no subscripts are given then all of the values are returned.

Example: Matrix Subscripting

```
> mat[]  #Blank
      col a col b col c
row a  -2.0  -0.4   1.2
row b  -1.2   0.4   2.0
> mat[2,2]  # getting an element
[1] 0.4
> mat[1,]  # getting a row
 col a col b col c
    -2  -0.4   1.2
```

```
> mat[,1]  # getting a column
 row a row b
    -2  -1.2
> mat[c(F,T),]  #logical subscript
 col a col b col c
  -1.2   0.4     2
> mat["row a","col b"] #by dimention name
[1] -0.4
```

Factors and Ordered Factors

- They provide a way to represent categorical vectors.
- When using Ordered Factors we are assuming some sort of natural ordering (e.g. Low, High)
- Most of the advanced statistical modeling functions such as lm, aov, glm, gam, loess and nls recognize that **Factors** represent categorical variables and treat them appropriately

- Ordered Factors include information about the ordering of the categories. So, they are treated differently than factors by the modeling functions.
- A Factor is stored as a vector of integers corresponding to the possible categories. The names of the categories are stored as levels attribute of the factor.
- Factors and Ordered factors have the following main attributes: levels, class
  - Levels: gives the levels of values and can be displayed with the **level** function.
  - class: specifies whether it is a Factor or an Ordered Factor and can be displayed with the **class** function.

Creating Factors

- Use the **factor** function to create a factor
  - **levels** argument can be used to specify the levels
  - **labels** argument can be used to specify labels for the levels

Example: Creating Factors

```
> color <- factor(c("Red","Green","Yellow","Red"))
> color
[1] Red    Green  Yellow Red

> color <- factor(c("Red","Green","Yellow","Red"),
+                 levels=c("Red","Green","Yellow"),
+                 labels=c("Red Color",
+                 "Green Color","Yellow Color"))
> color
[1] Red Color    Green Color  Yellow Color
[4] Red Color
```

Example: Factors Attributes

```
> levels(color)
[1] "Green"  "Red"    "Yellow"
> class(color)
[1] "factor"
```

Creating Ordered Factors

- Use the **ordered** function to create an ordered factors
  - **levels** argument can be used to specify the levels
  - The order of the categories in the levels attribute specifies the ordering of the levels
  - **labels** argument can be used to specify labels for the levels

Example: Creating Ordered Factors

```
> education <- ordered(c("Hi","Hi","Med","Lo",
+              "Lo","Med"),
+              levels=c("Lo","Med","Hi"))
> education
[1] Hi  Hi  Med Lo  Lo  Med

 Lo < Med < Hi
```

Example: Ordered Factors Attributes

```
> levels(education)
[1] "Lo"  "Med" "Hi"
> class(education)
[1] "ordered" "factor"
```

Non-Atomic Data Objects

Data Frames

- They are data objects that allows you to group data by variables (columns) regardless of their type. Note that all of the variables must be of the same length.
- The attributes of a data frame are length, mode, names, rownames and class.
  - Length gives the number of components
  - Mode is “list’.
  - Names gives the variable label.
  - row.names gives the row or observation labels.
  - Class shows that the object we have is a “dataframe’.

Creating Data frames

- Use **data.frame** function to combine vectors into data frame
- Use **as.data.frame** function to convert a matrix to data frame

Example: date.frame function

```
> name <- c("Tom","Susan","Fay","Sam")
> grade <- c(90,70,80,65)
> class.frame <- data.frame(name,grade)
> class.frame
   name grade
1   Tom    90
2 Susan    70
3   Fay    80
4   Sam    65
```

Example: as.date.frame function

```
> mat.to.frame <- as.data.frame(mat)
> mat.to.frame
      col a col b col c
row a  -2.0  -0.4   1.2
row b  -1.2   0.4   2.0
```

Example: attributes of Data Frame

```
> mode(class.frame)
[1] "list"
> names(class.frame)
[1] "name"  "grade"
> row.names(class.frame)
[1] "1" "2" "3" "4"
> class(data.frame)
NULL
> class(class.frame)
[1] "data.frame"
```

Data Frame Subscripting

- Data Frame could be subscribed as a
  - matrix
  - list

Example: Data Frame Subscripting

```
> class.frame[1,]
  name grade
1  Tom    90
> class.frame[1:2,]
   name grade
1   Tom    90
2 Susan    70
> class.frame[,c(T,F)]
[1] Tom   Susan Fay   Sam
> class.frame$grade
[1] 90 70 80 65
```

Lists data Objects

- It is an ordered collection of components.
- Each components can be any data object.
- Different list component can be of different modes.
- List Attributes are length, mode and names
  - Length gives the number of components
  - Mode is “list’.
  - Names gives the name for each component

Creating Lists

- Use the **list** function to combine objects into a List

Example: Creating List

```
> list1 <- list(x=x,mat=mat)
> list1
$x:
[1] 2 3 4 5

$mat:
      col a col b col c
row a  -2.0  -0.4   1.2
row b  -1.2   0.4   2.0
```

Example: List Attributes

```
> length(list1)
[1] 2
> mode(list1)
[1] "list"
> names(list1)
[1] "x"   "mat"
```

Subscripting Lists

- Use the $ operator with the name of the component.
- Put the number of the component in double square brackets [[ ]].
- Put the name of the component surrounded in quotes in double square brackets [[ ]].

Example: List Subscript

```
> list1$mat
       col a col b col c
row a  -2.0  -0.4   1.2
row b  -1.2   0.4   2.0
> list1[[1]]
[1] 2 3 4 5
> list1[["x"]]
[1] 2 3 4 5
```

Exercises:Data Objects

Q1. What is the difference between the following two lines?

```
> Letters <- c(T,F)
> Letters <- c(“T”,”F”)
```

Q2. Do the following

```
> num1 <- “2”
> num2 <- “3”
> num1+num2
```

What do you get?

Q3. Change the mode of num1 and num2 to numeric? Then do

```
>num1+num2
```

What do you get now?

Q4. a. Find the values of the following:

```
> rep(5,1)
> rep(1,5)
> rep(c(0,6),2)
> rep(c("a","b"),3)
> rep(1:3,4)
> rep(c(1,5,8),length=10)
> rep(1:5,1:5)
```

b. Assign rep(1:5,1:5) to the name x? Find the length of x using the **length** function.

c. Force the length of x to be 5 by doing

```
> length(x) <- 5
```

What happened?

Q5. Find the values of the following:

```
> seq(1,5,1)
> seq(1,5)
> seq(5)
> seq(5,1,-1)
> seq(5:1)
> 5:1
> seq(3,4,0.1)
```

Q6. Let

```
> x <- seq(1,15,1)
> y <- x > 5
> z <- x[x>5]
```

What are the values of y and z?

Q7. Let

```
> grade <- c(5,7,8)
> name <- c("Sam","Dan","Susan")
```

a. Do,

```
> c(grade,name)
```

What do you notice?

b. Do,

```
> c(grade,T,T,F,T)
```

What do you notice?

Q8. Make a matrix that looks like this

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

And like this one

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Q9. a. Using the rbind function create a matrix from the following vectors

```
> grade1 <- c(5,7,8)
> grade2 <- c(6,6,10)
```

b. What is the mode for this matrix?

c. What is the dim of this matrix?

d. What is the dimnames of this matrix?

e. Use the following vector for column names

   name <- c("Sam","Dan","Susan")

f. Write out the first column of the matrix.

g. Write out the rows for Sam and Susan.

h. Use the function **t** to find the transpose of this matrix.

Q10. a. Create a data frame that looks like the following:

```
> authors
   FirstName  LastName  Age   Income        Home
1      Lorne     Green   82  1200000  California
2      Loren      Jaye   40    40000  Washington
3      Robin     Green   45    25000  Washington
4      Robin      Howe    2        0     Alberta
5      Billy      Jaye   40    27500  Washington
```

b. What is the out come of the following two lines:

```
> authors$Age[1:3]
> authors[1:3]$Age
```

Q11. For the built in object **air**

a. Show that it is a data frame

b. Find the name of the variables

c. Find the row names

Q12. a. Create the following two lists

```
> list1 <- list(a=1:3,b=rep(4,5),l=letters[1:5])
> list2 <- list(list1=list1,old=list(1:5,7:4),c(2,3))
```

c. Print the names of the two lists using the names function.

d. Find the values of these

```
list1$b          list2$list1$l          list2[[3]]
list2[[3]]+4  list2[3]+4
```

Q13. We can use the **cut** function to create factor objects.

a. Type the following and look at the result.

```
> age <- c(27,80,5,35,45,10,77,68)
> age.factor <- factor(cut(age,
  + breaks=c(0,30,60,90)),
  + levels=c(1,2,3),
  + label=c("Young","Middle Age","Old"))
```

b. Find the attributes of age.factor

Statistical Models

Contents:

- Linear Regression

Linear Regression

Regression Methods in Splus

Function	**Explanation**
aov	ANOVA
gam	Generalized additive model
glm	Generalized linear model
lm	Linear models
loess	Local regression analysis
nls, ms	Non-linear models
tree	Tree-based models

Specifying Formulas

Expression	**Meaning**
T ~ F	T is predicted linearly in F
F_a + F_b	Include both F_a and F_b in the model
F_a – F_b	Include all of F_a except what is in F_b in the model
F_a: F_b	The interaction between $F_a$and F_b
F_a* F_b	F_a + F_b + F_a: F_b
F_a %in% F_b	F_b is nested within F_a
F_a/ F_b	F_a + F_a %in% F_b
F^m	All terms in F crossed to order m

Example:

- Consider the built-in data sets ***stack.loss*** and ***stack.x***. Together, the two datasets contains information on ammonia loss in a manufacturing process.
- ***stack.x***: is a is a matrix that contains the three columns air flow, water temperature and acid concentration which are going to be considered the predictors.
- ***stack.loss***: is a is a vector containing the percent of ammonia lost times 10 and is going to be considered the response.

Steps:

- Organizing the data
- Exploratory data analysis
- Fitting the model
- Producing summaries and diagnostic plots
- Fitting an alternative models
- Comparing the fitted models for goodness of fit
- Prediction
- Using the selected model.

Step 1: *Organizing the Data*

- Combine the two datasets into a data frame

```
> stack.df <- data.frame(stack.loss,stack.x)
> names(stack.df)
[1] "stack.loss" "Air.Flow"   "Water.Temp"
    "Acid.Conc.“
```

- Attach the data frame stack.df

```
> attach(stack.df)
```

Step 2: *Exploring the Data*

- To get a summary statistics of the data use the **summary** function.

```
> summary(stack.df)
   stack.loss       Air.Flow      Water.Temp      Acid.Conc.
   Min.: 7.00      Min.:50.00      Min.:17.0      Min.:72.00
 1st Qu.:11.00   1st Qu.:56.00   1st Qu.:18.0   1st Qu.:82.00
  Median:15.00    Median:58.00    Median:20.0    Median:87.00
   Mean:17.52      Mean:60.43      Mean:21.1      Mean:86.29
 3rd Qu.:19.00   3rd Qu.:62.00   3rd Qu.:24.0   3rd Qu.:89.00
   Max.:42.00      Max.:80.00      Max.:27.0      Max.:93.00
```

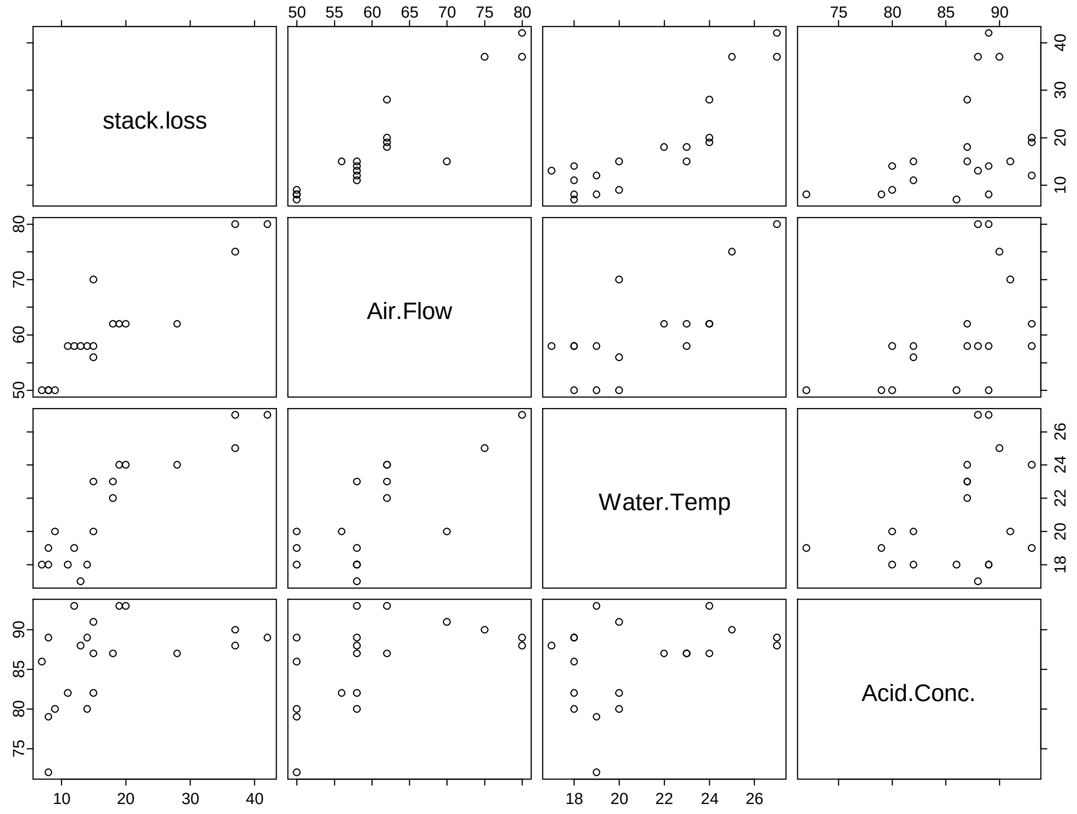
- To get the correlation matrix use the **cor** function.

```
> cor(stack.df)
            stack.loss  Air.Flow Water.Temp Acid.Conc.
stack.loss   1.0000000 0.9196635  0.8755044  0.3998296
Air.Flow     0.9196635 1.0000000  0.7818523  0.5001429
Water.Temp   0.8755044 0.7818523  1.0000000  0.3909395
Acid.Conc.   0.3998296 0.5001429  0.3909395  1.0000000
```

So as we can see the Air.flow and Water.Temp are both highly correlated and both are highly correlated with stack.loss.

- To produce a scatterplot matrix use the **pairs** function.

```
> pairs(stack.df)
```



We can see that there is a strong linear relationship between stack.loss and both Air.flow and Water.Temp.

Also, the two predictor variables are themselves correlated.

Step 3: *Fitting a model*

- To fit a multiple regression model predicting stack.loss by all three covariates. We do the following

```
> fit1 <- lm(stack.loss ~ Air.Flow + Water.Temp + Acid.Conc.)
```

- To get the estimates do the following:

```
> fit1
Call:
lm(formula = stack.loss ~ Air.Flow + Water.Temp +
    Acid.Conc.)

Coefficients:
 (Intercept)  Air.Flow Water.Temp Acid.Conc.
   -39.91967 0.7156402   1.295286 -0.1521225

Degrees of freedom: 21 total; 17 residual
Residual standard error: 3.243364
```

- To get more detailed description of the model use the **summary** function:

```
> summary(fit1)
Call: lm(formula = stack.loss ~ Air.Flow + Water.Temp + Acid.Conc.)
Residuals:
   Min     1Q  Median    3Q   Max
 -7.238 -1.712 -0.4551 2.361 5.698

Coefficients:
                     Value Std. Error  t value Pr(>|t|)
(Intercept )     -39.9197  11.8960  -3.3557   0.0038
 Air.Flow         0.7156   0.1349     5.3066   0.0001
 Water.Temp   1.2953   0.3680     3.5196   0.0026
 Acid.Conc.    -0.1521   0.1563    -0.9733   0.3440
```

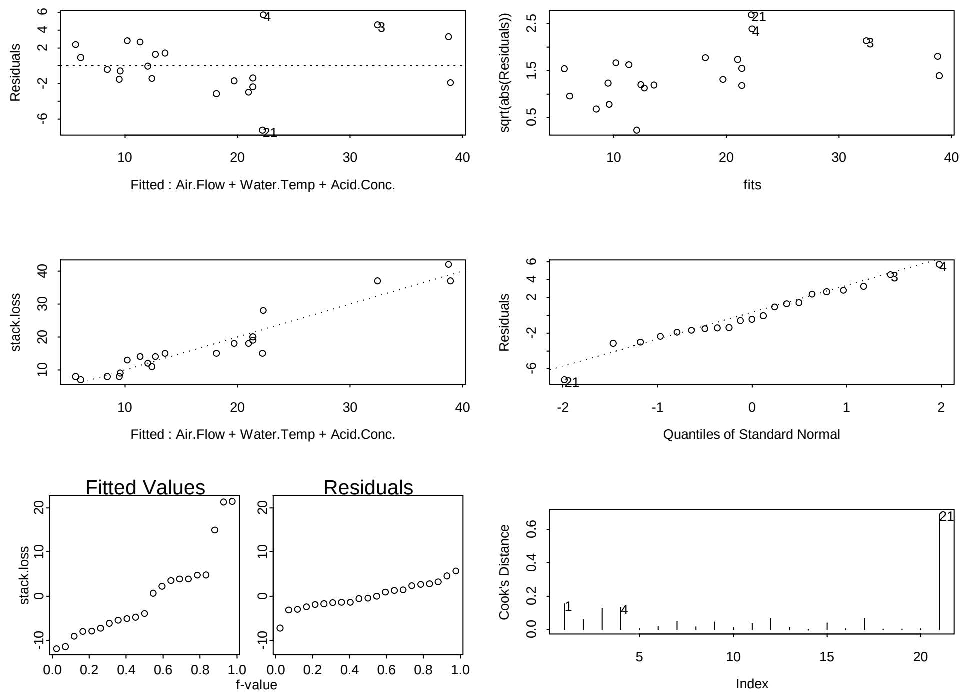
```
Residual standard error: 3.243 on 17 degrees of freedom
Multiple R-Squared: 0.9136
F-statistic: 59.9 on 3 and 17 degrees of freedom, the p-value is 3.016e-009

Correlation of Coefficients:
            (Intercept) Air.Flow Water.Temp
Air.Flow      0.1793
Water.Temp -0.1489     -0.7356
Acid.Conc.  -0.9016     -0.3389   0.0002
```

As we can see Air.Flow and Water.Temp are important predictors of stack.loss.

Step 4: *Producing Diagnostic Plots*

- To produce a set of diagnostic plots we can use the function **plot.lm**.

```
> par(mfrow=c(3,2))
> plot.lm(fit1)
```



Step 5: *Fitting Alternative Models*

- Since the t-value for Acid.Conc. is small, it suggests that this variable does not add to the model so it could be deleted.

```
> fit2 <- update(fit1,. ~ . - Acid.Conc.)
> fit2
Call:
lm(formula = stack.loss ~ Air.Flow + Water.Temp)
Coefficients:
 (Intercept)  Air.Flow Water.Temp
   -50.35884 0.6711544   1.295351
Degrees of freedom: 21 total; 18 residual
Residual standard error: 3.238615
```

```
> summary(fit2)
Call: lm(formula = stack.loss ~ Air.Flow + Water.Temp)
Residuals:
    Min    1Q Median    3Q   Max
 -7.529 -1.75 0.1894 2.116 5.659
Coefficients:
               Value Std. Error  t value Pr(>|t|)
(Intercept) -50.3588   5.1383    -9.8006   0.0000
   Air.Flow   0.6712   0.1267     5.2976   0.0000
 Water.Temp   1.2954   0.3675     3.5249   0.0024
Residual standard error: 3.239 on 18 degrees of freedom
Multiple R-Squared: 0.9088
F-statistic: 89.64 on 2 and 18 degrees of freedom, the p-value is 4.382e-010
Correlation of Coefficients:
           (Intercept) Air.Flow
  Air.Flow -0.3104
Water.Temp -0.3438     -0.7819
```

Step 6: *Comparing Models*

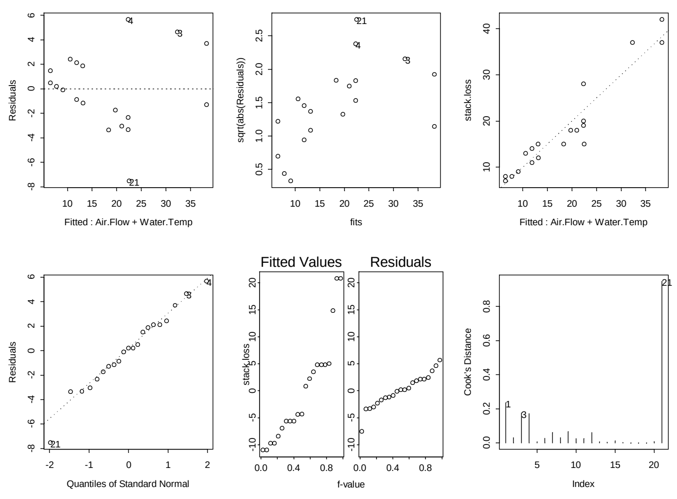
- To compare the two models we use the function **anova** as follows:

```
>anova(fit1,fit2)
Analysis of Variance Table
Response: stack.loss
                              Terms Resid. Df      RSS        Test Df Sum of Sq
1 Air.Flow + Water.Temp + Acid.Conc.        17 178.8300
2             Air.Flow + Water.Temp        18 188.7953 -Acid.Conc. -1 -9.965372
      F Value     Pr(F)
1
2      0.9473319 0.3440461
```

The multiple R2 has decreased only from 0.9136 to 0.9088. Also the F-value = 0.9473319 and Pr(F) = 0.3440461. Which suggest that the larger model does not predict stack.loss significantly better than the smaller model. So, the smaller model is preferred.

The diagnostic plots for fit2 also suggest that the model is appropriate.

```
> par(mfrow=c(2,3))

> plot(fit2)
```



Step 7: *Prediction*

To get predicted values of new predictor values, we could use the **predict** function.

```
> new.values <- data.frame(Air.Flow=c(55,80),
+ Water.Temp=c(16,26))
> predict(fit2,new.values)
       1        2
 7.280276 37.01265
```

Step 8: *Using the model*

In S_PLUS we can subscript an object of linear model fit like fit2 or the result of summary(fit2) to obtain components of the fit as S-PLUS objects.

```
> names(fit2)
 [1] "coefficients"  "residuals"     "fitted.values" "effects"       "R"
 [6] "rank"          "assign"        "df.residual"   "contrasts"     "terms"
[11] "call"
> fit2$coef
 (Intercept)  Air.Flow Water.Temp
   -50.35884 0.6711544   1.295351
> names(summary(fit2))
 [1] "call"          "terms"         "residuals"     "coefficients" "sigma"
 [6] "df"            "r.squared"     "fstatistic"    "cov.unscaled" "correlation"
> summary(fit2)$df
[1]  3 18  3
```

Or, you could use the extractor functions **coef, resid and fitted** which returns the coefficient, residuals and fitted values respectively.

```
> coef(fit2)
 (Intercept)  Air.Flow Water.Temp
   -50.35884 0.6711544   1.295351
> resid(fit2)
       1        2        3        4        5         6         7         8
3.691998 -1.308002 4.638473 5.658832 -1.750465 -3.045817 -3.341168 -2.341168
        9       10       11       12         13        14       15        16
-3.361199 2.115558 2.115558 2.410909 -0.8844421 -1.179793 1.484793 0.4847934
      17       18         19       20        21
0.189442 0.189442 -0.1059093 1.867164 -7.528998
> fitted(fit2)
     1      2        3        4        5        6        7        8       9
38.308 38.308 32.36153 22.34117 19.75047 21.04582 22.34117 22.34117 18.3612
      10       11       12       13       14       15       16       17       18
11.88444 11.88444 10.58909 11.88444 13.17979 6.515207 6.515207 7.810558 7.810558
      19       20     21
9.105909 13.13284 22.529
```

Exercises:Statistical Models

Q1.a. Create a data frame called swiss.df from the matrix swiss.x and the vector swiss.fertility.

b. Get a summary of swiss.df.

c. Get the correlation coefficient between all of the variables.

d. Produce a scatter plot matrix for all of the variables.

e. Fit a model with both Education and Catholic as predictors and swiss.fertility as the response.

f. Fit a model with Education as a predictor and swiss.fertility as the response.

g. Fit a model with Catholic as a predictor and swiss.fertility as the response.

h. Determine whether Education alone, Catholic alone or both should be included in the model.

Graphics

Contents:

Introduction

- It is one of the most widely used mediums of transferring information about data.
- It allows researchers to visualize data in order to detect interesting features or structure in the data.

Graphic Devices

- Two kinds of graphic devices
  - Graphic Windows
  - Printing Devices
- To start a graphsheet without plotting type
  - >graphsheet()
- To start a printer device without plotting
  - >graphsheet(format="printer")
- To close the graphics device type
  - >dev.off() (note that a printer device must be closed to send the graph to the printer or the specified file)

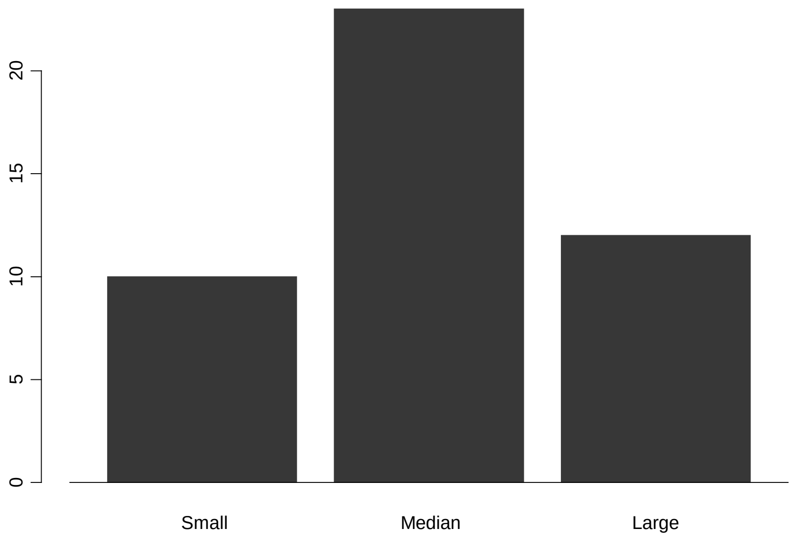
- You can have several active graphic devices.
- Only one graphic devices is the current graphic device.
- **dev.list()** tells what graphic devices are active
- **dev.cur()** tells which graphic device is the current one.
- You can change the current device by using **dev.set(x)** where x is the graphic device that you want to be the current.
- To turn off a graphic device use **dev.off()**.

Graphic Functions: *Univariate Data*

Function	**Explanation**
barplot	Simple bar plot
boxplot	Simple box plot
hist	Histogram
dotchart	Dot chart
pie	Pie chart
qqnorm	Quantile-Quantile plot for a sample against the standard normal

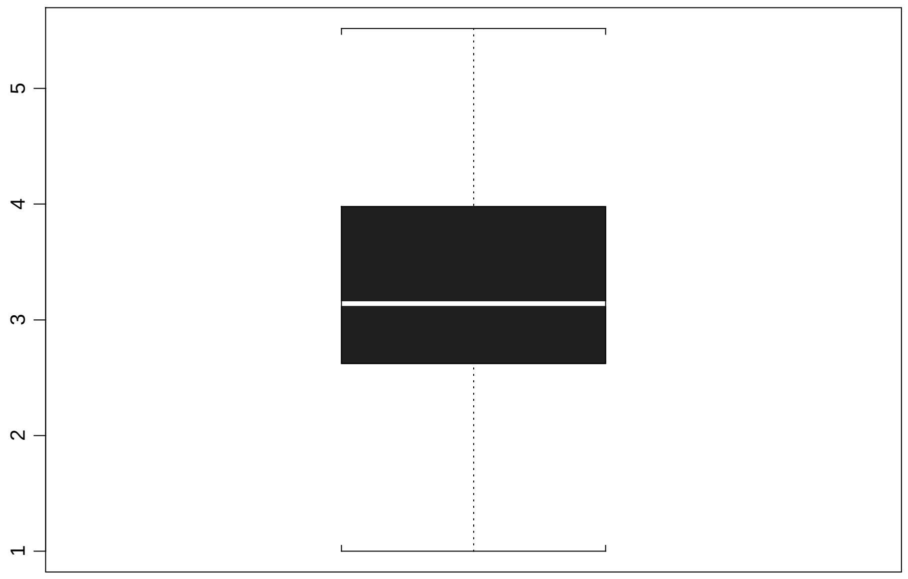
Example:

```
> x <- c(10,23,12)
> names(x) <- c("Small","Median","Large")
> x
 Small Median Large
    10     23    12
> barplot(x, names=names(x))
```



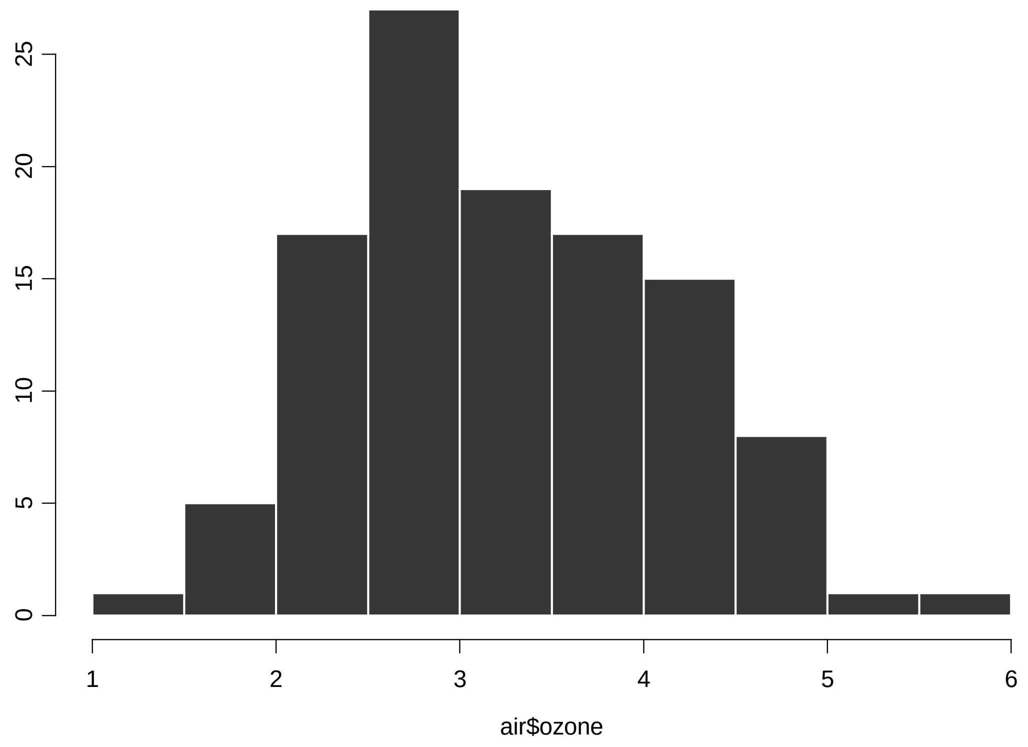
Example:

> boxplot(air$ozone)



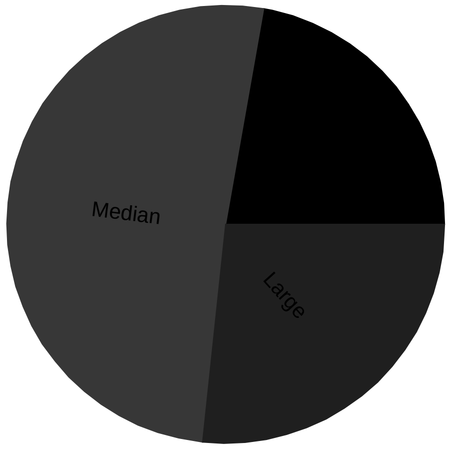
Example:

```
> hist(air$ozone)
```



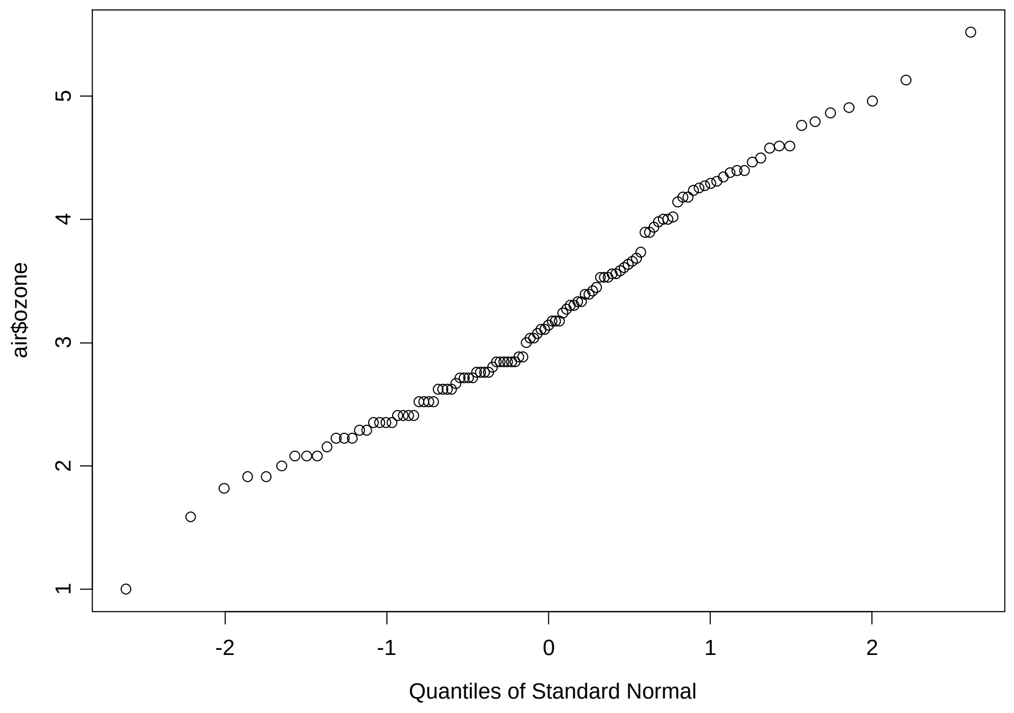
Example:

>pie(x,names=names(x))



Example:

```
> qqnorm(air$ozone)
```

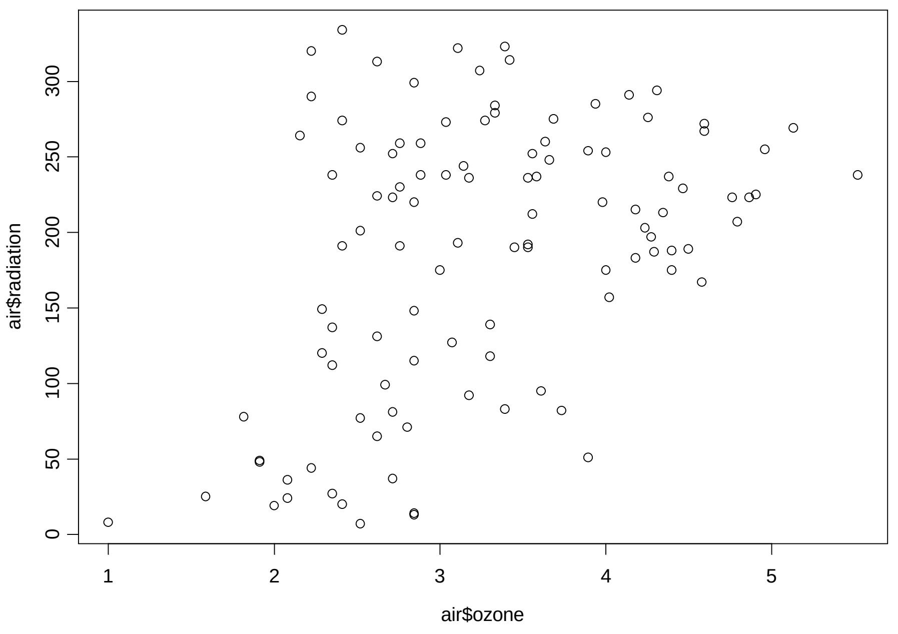


Graphic Functions: *Bivariate Data*

Function	**Explanation**
plot	Scatterplot
barplot	Simple bar plot
boxplot	Side-by-side box plots
qqplot	Quantile-quantile plot for two samples

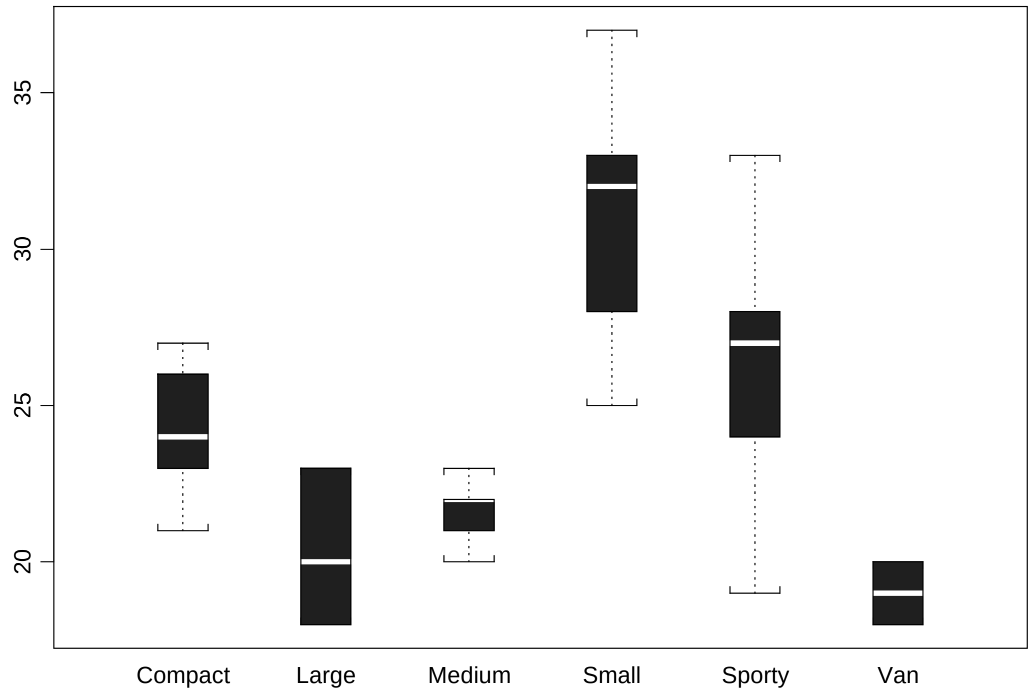
Example:

- > plot(air$ozone,air$radiation)



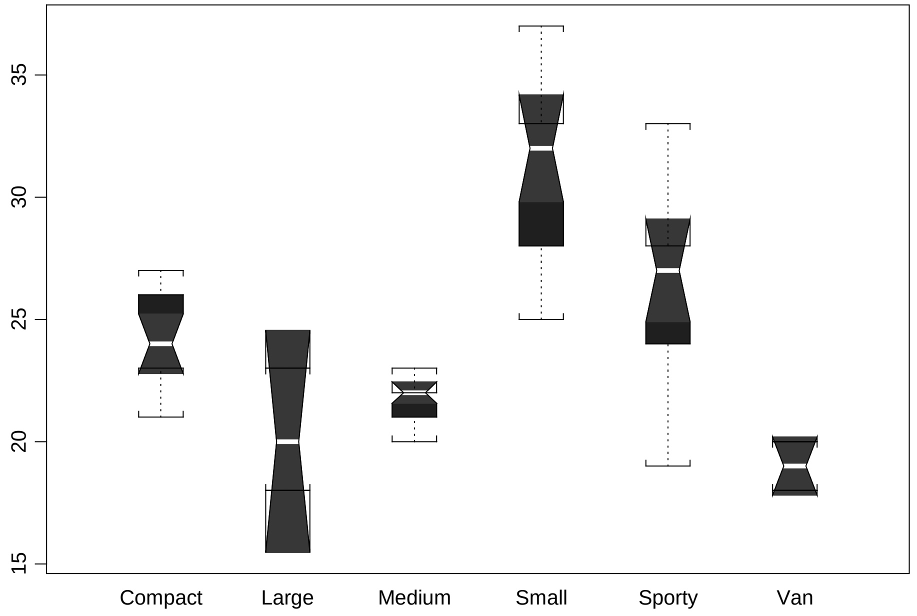
Example:

```
> boxplot(split(fuel.frame$Mileage,fuel.frame$Type))
```



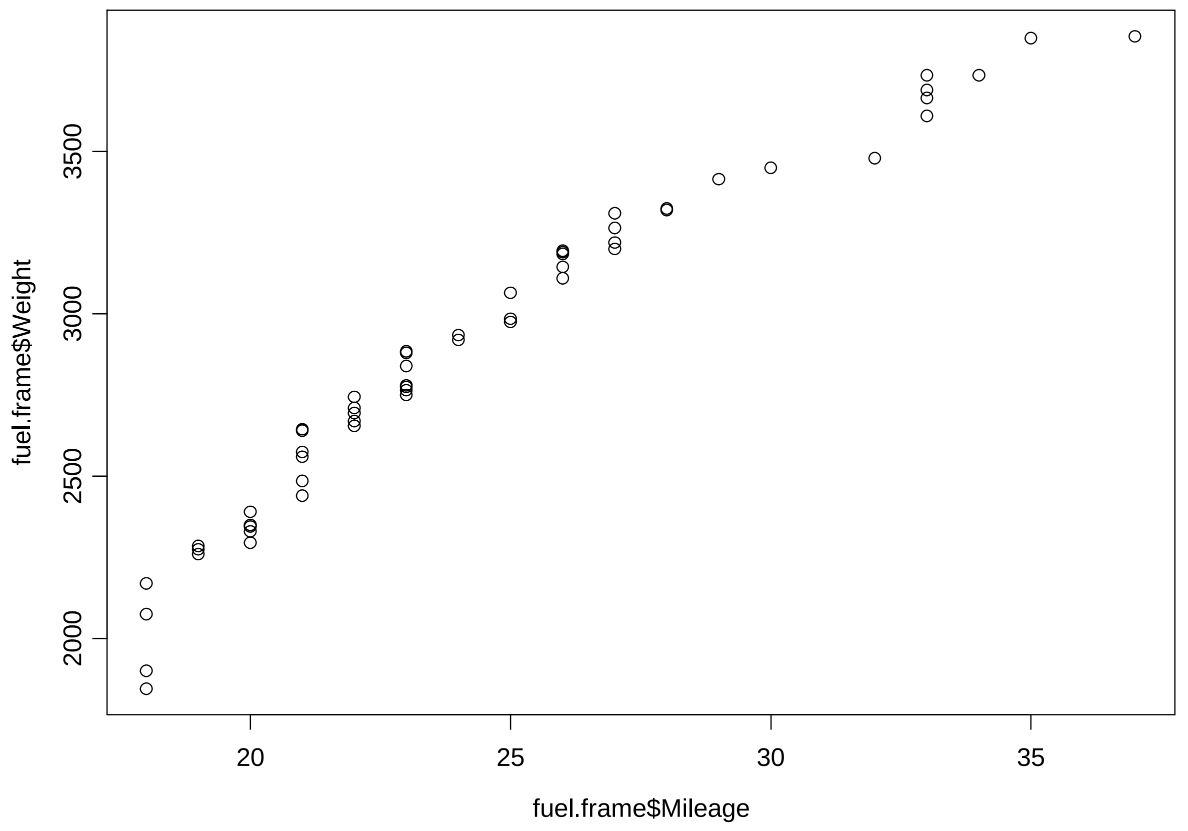
Example:

```
> boxplot(split(fuel.frame$Mileage,fuel.frame$Type),notch=T)
```



Example:

```
> qqplot(fuel.frame$Mileage,fuel.frame$Weight)
```



Graphic Functions: *Three-Dimensional Plots*

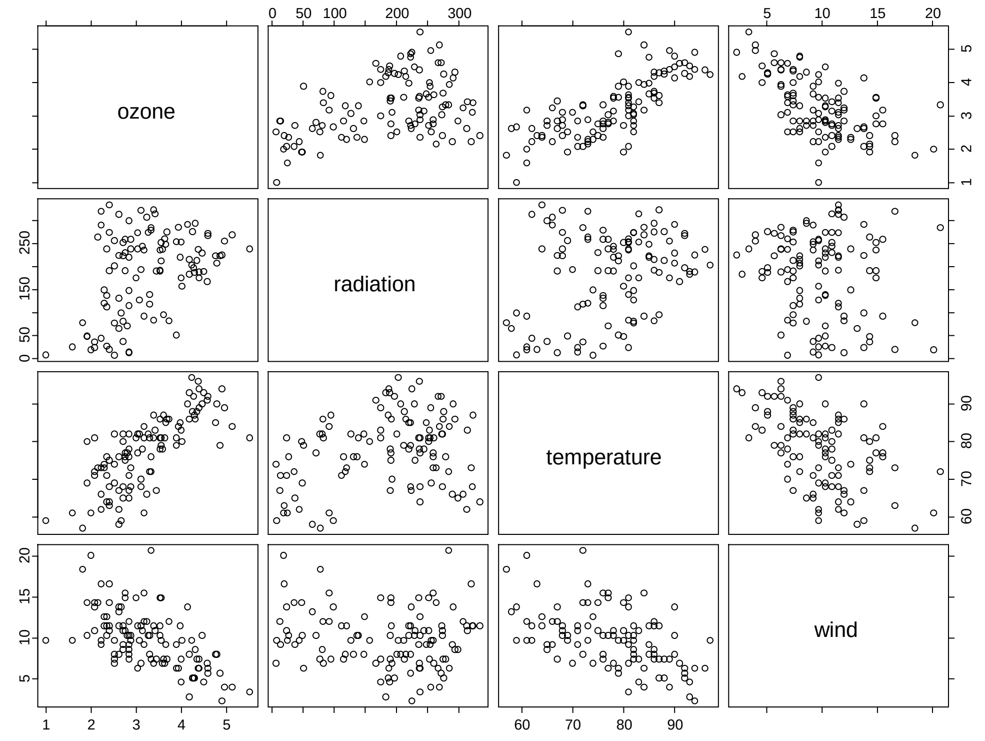
Function	**Explanation**
contour	Contour plot
persp	Perspective plot
image	Color or grayscale image plot

Graphic Functions: *Multivariate Data*

Function	**Explanation**
pairs	Pairwise scatter plot matrix
coplot	Scatterplots conditioned on a third variable
symbols	Scatterplot with symbols determined by third variable

Example:

```
> pairs(air)
```

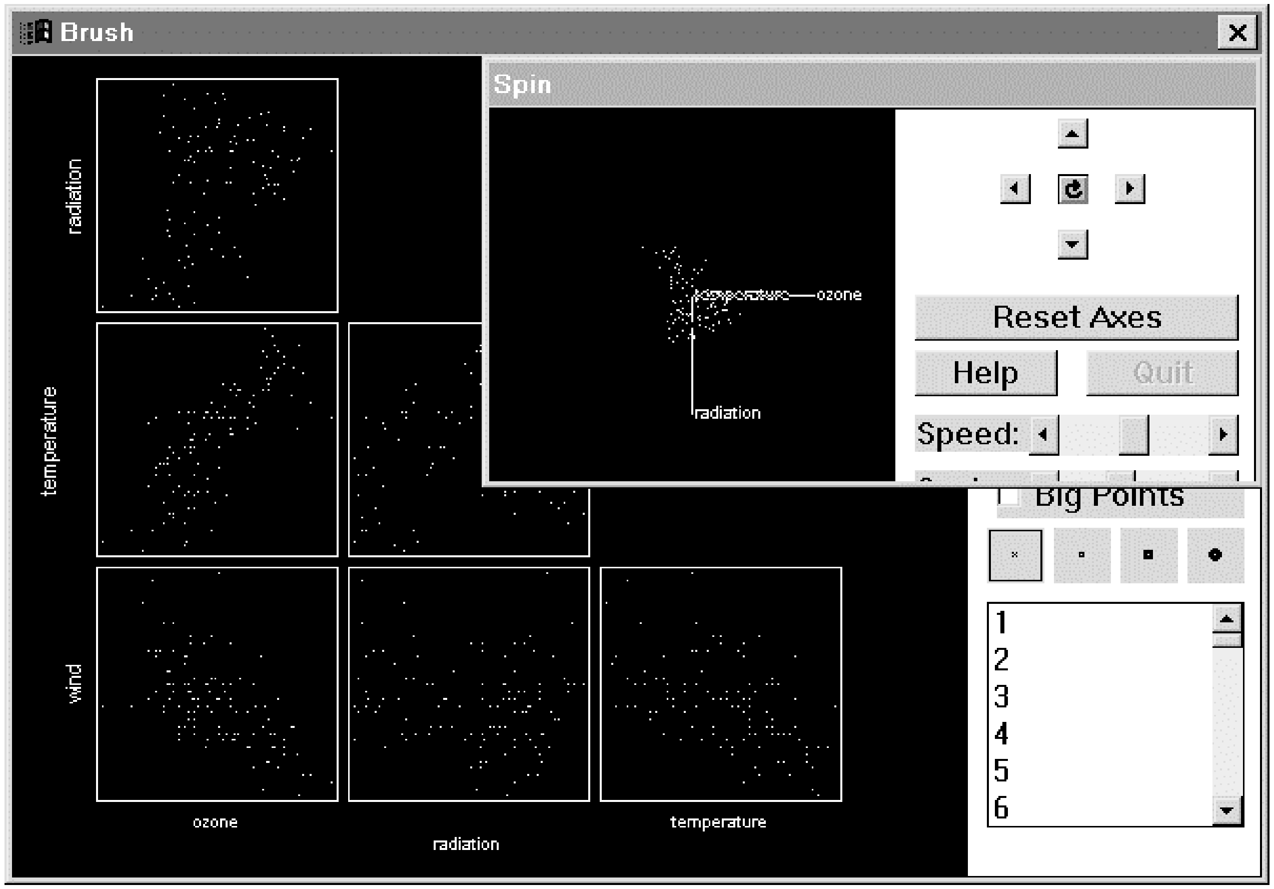


Graphic Functions: *Dynamic Graphics*

Function	**Explanation**
brush	Create linked scatterplots and rotatable point cloud

Example:

> brush(air)



Multiple Plot Layout

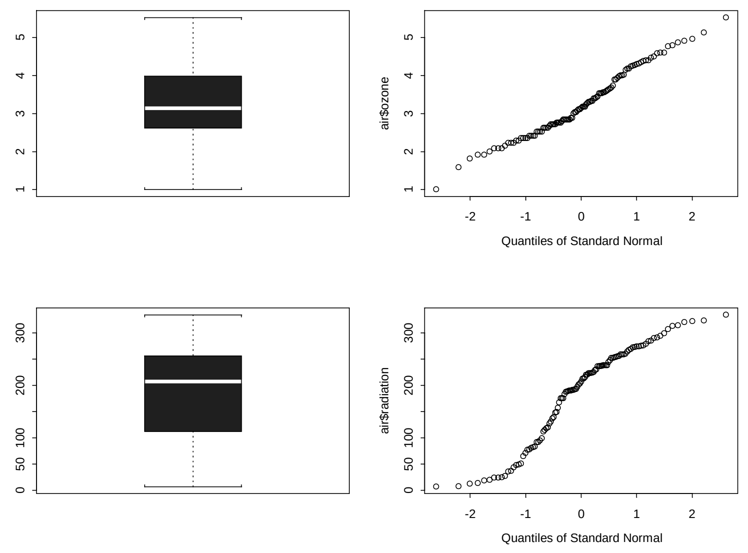
- To display more than one plot on a single page, use the "**par**" function
- Syntax
  - par(mfrow=c(n,m))

    Where n = number of rows

    m = number of columns

Example:

```
> par(mfrow=c(2,2))
> boxplot(air$ozone)
> qqnorm(air$ozone)
> boxplot(air$radiation)
> qqnorm(air$radiation)
```



Plot Shape

- The default shape of the box enclosing the plot is *rectangular*.
- To change it to a square box at the S-Plus prompt we write

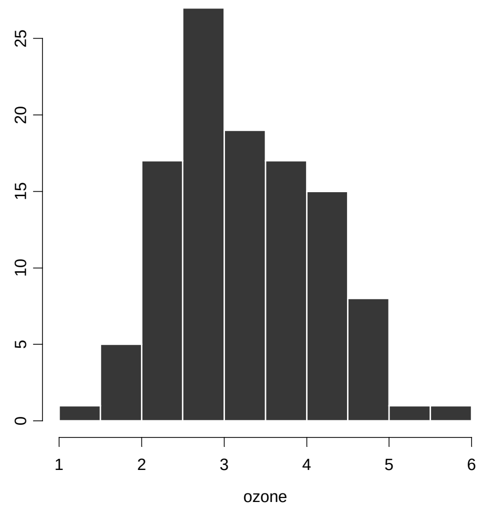
  ```
  > par(pty=“s”)
  ```

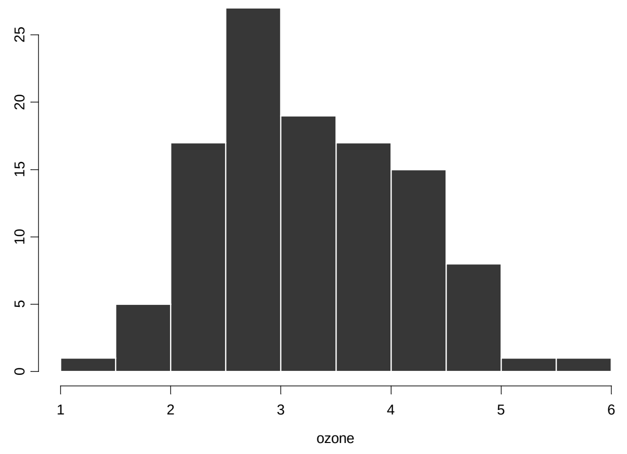
  where pty stands for “plot type”
- To change it back to the default at the S-Plus prompt we write

  ```
  > par(pty=“”)
  ```

Example:

```
> par(pty="s")
> hist(ozone)
> par(pty=“”)
> hist(ozone)
```



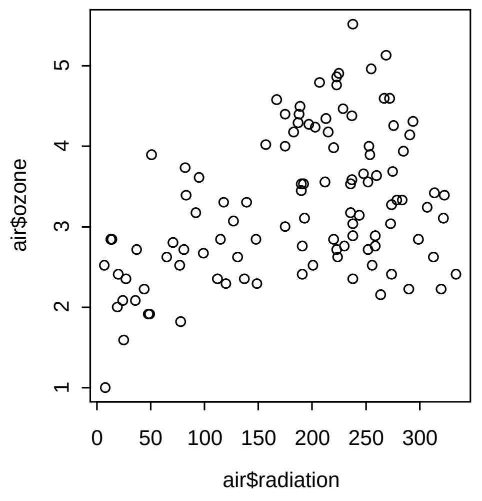


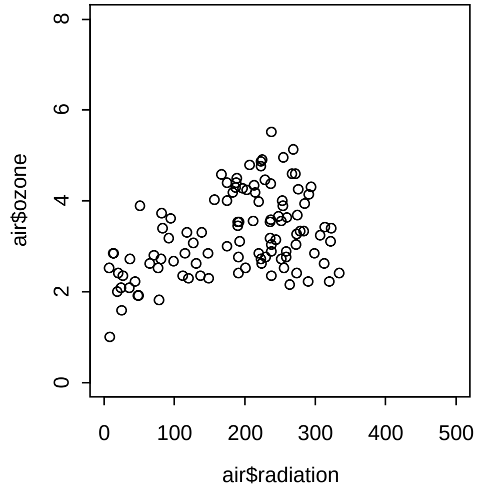
Commonly Graphic Parameters

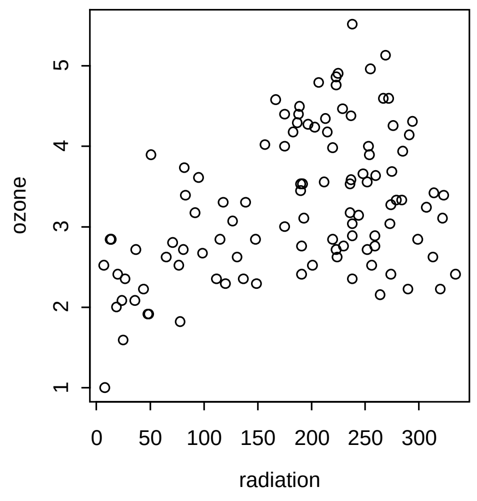
Argument	**Explanation**
xlim	Range of first (x) variable
ylim	Range of second (y) variable
pch	Plotting character
col	Color of points
lty	Line type
xlab	Label of first variable
ylab	Label of second variable
main	Main title at top of plot
sub	Subtitle under plot

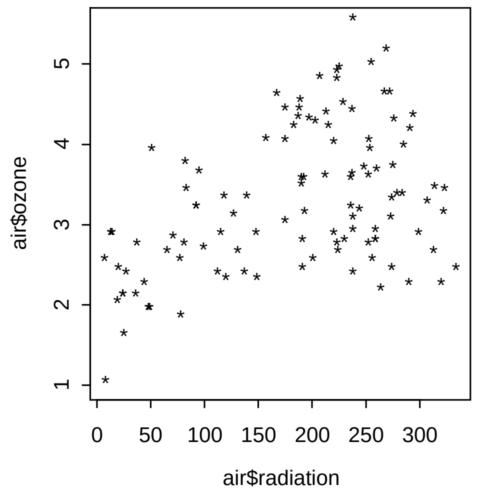
Example:

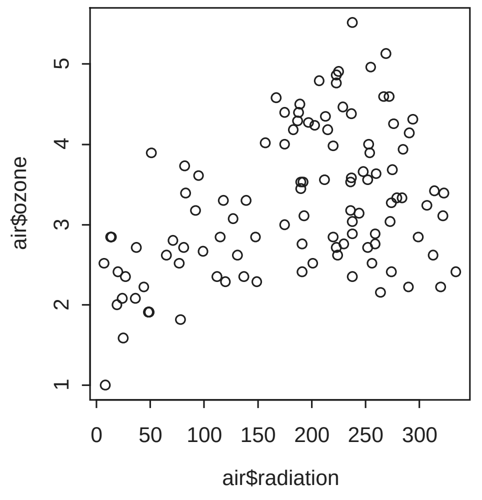
```
> par(mfrow=c(2,3))
> plot(air$radiation,air$ozone)
> plot(air$radiation,air$ozone,xlim=c(0,500),ylim=c(0,8))
>plot(air$radiation,air$ozone,xlab="radiation",ylab="ozone")
> plot(air$radiation,air$ozone,pch="*")
> plot(air$radiation,air$ozone,col=3)
> plot(air$radiation,air$ozone,main="Scatter Plot of radiation vs
   ozone")
```

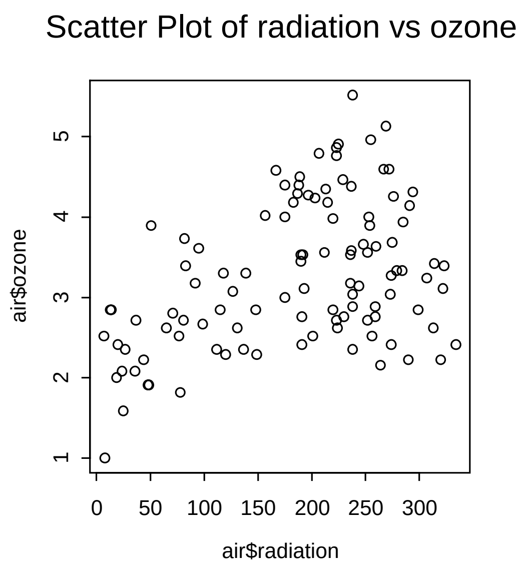

air$ozone
air$radiation


air$ozone
air$radiation


ozone
radiation


air$ozone
air$radiation


air$ozone
air$radiation

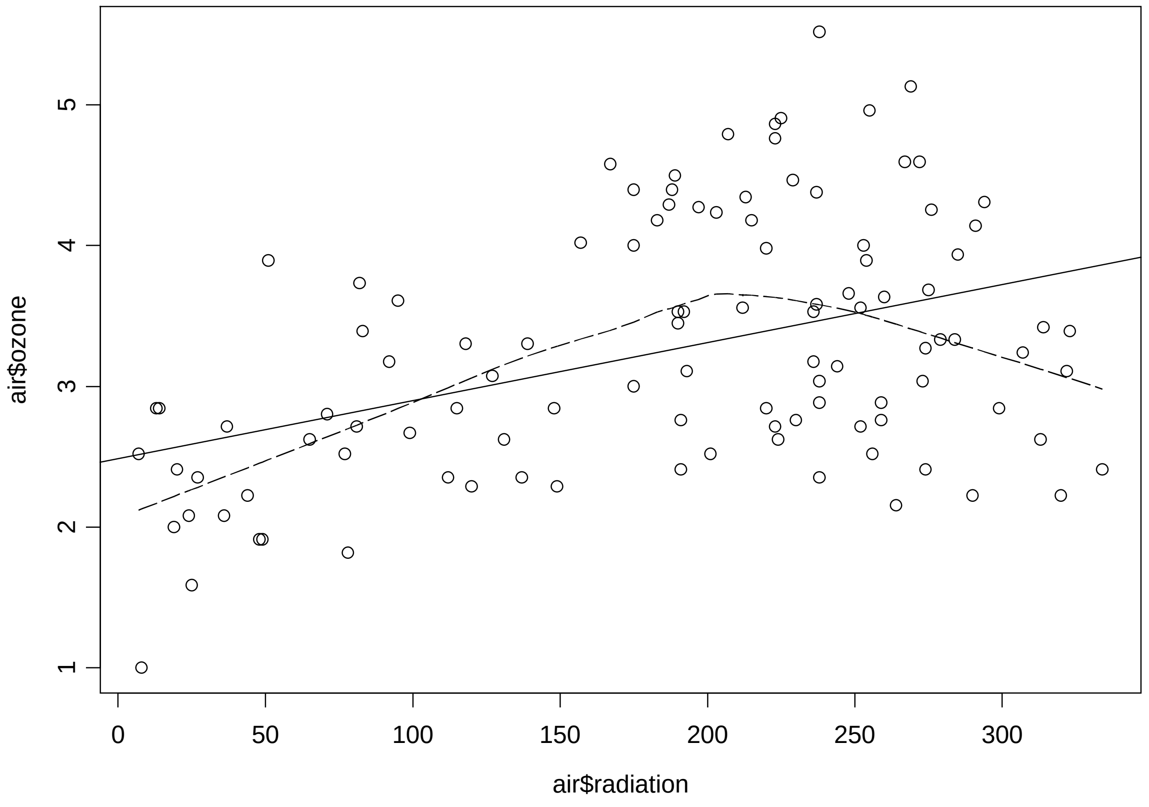

Scatter Plot of radiation vs ozone
air$ozone
air$radiation

Adding elements to an existing Plots

Function	**Explanation**
points	Add points
text	Add text
lines	Add lines connecting points
abline	Add straight line
lsfit	Fit least-square line
lowess	Sit lowess scatterplot smooth
title	Add titles
legend	Add legend
identify	Interactively identify points

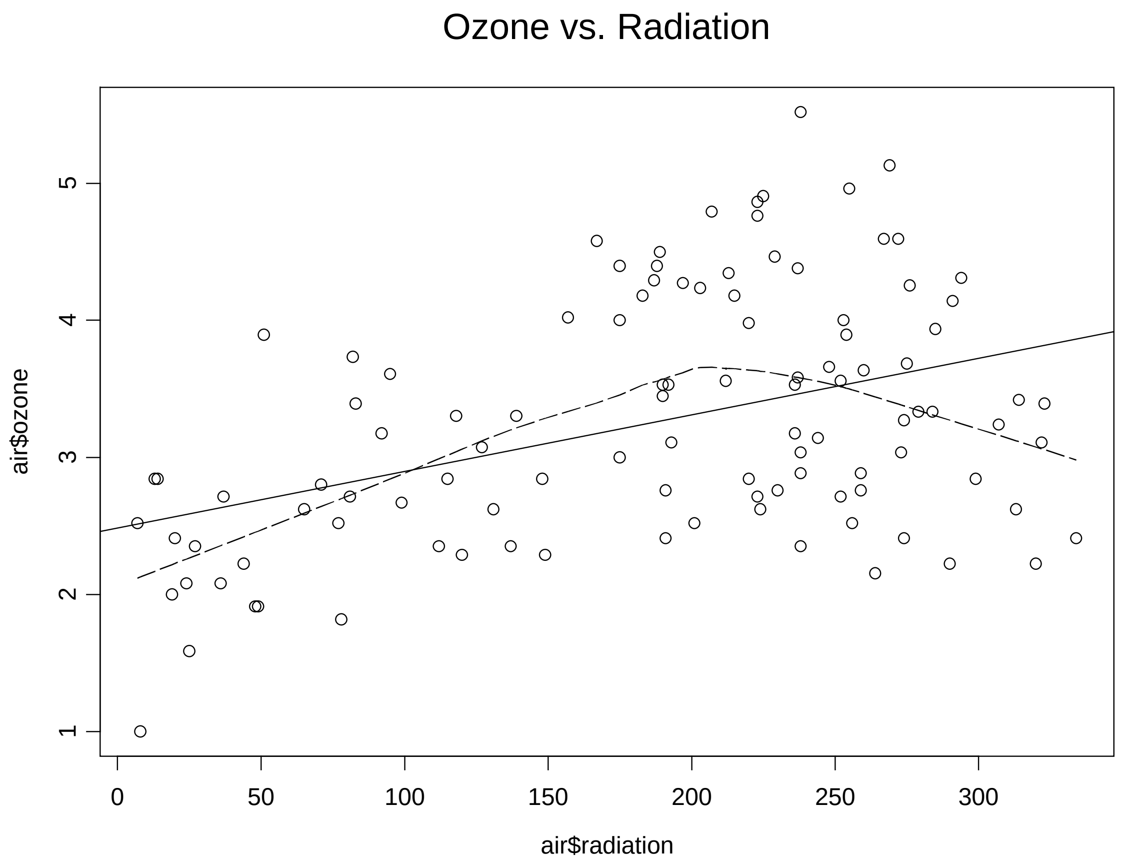
Example: Adding Lines

```
> plot(air$radiation,air$ozone)
> abline(lsfit(air$radiation,air$ozone))
> lines(lowess(air$radiation,air$ozone),lty=4)
```



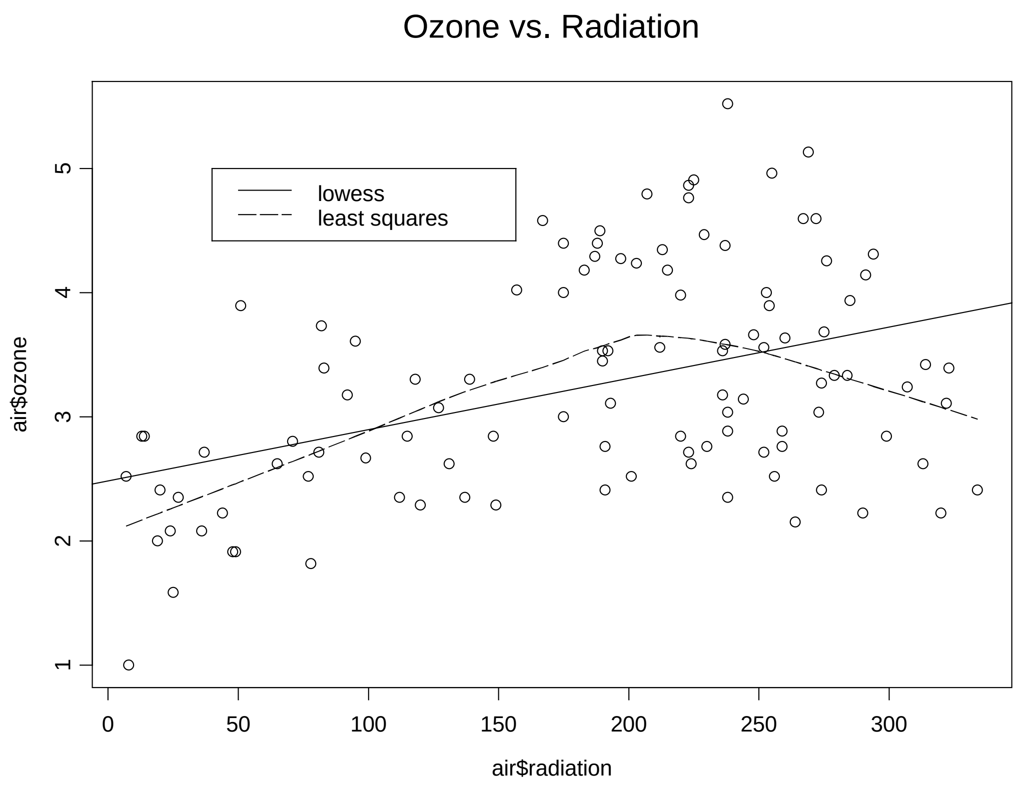
Example: Adding Titles

```
> title("Ozone vs. Radiation")
```



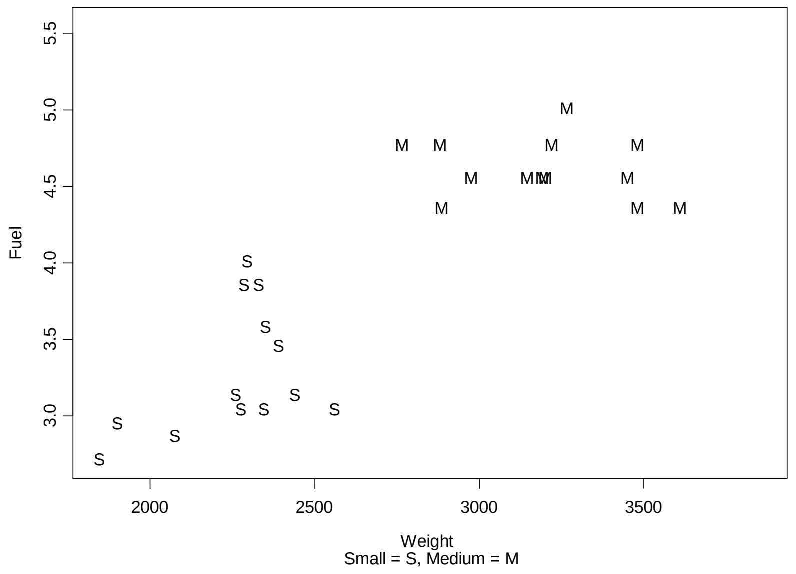
Example: Adding Legend

```
> legend(40,5,c("lowess","least squares"),lty=c(4,1))
```



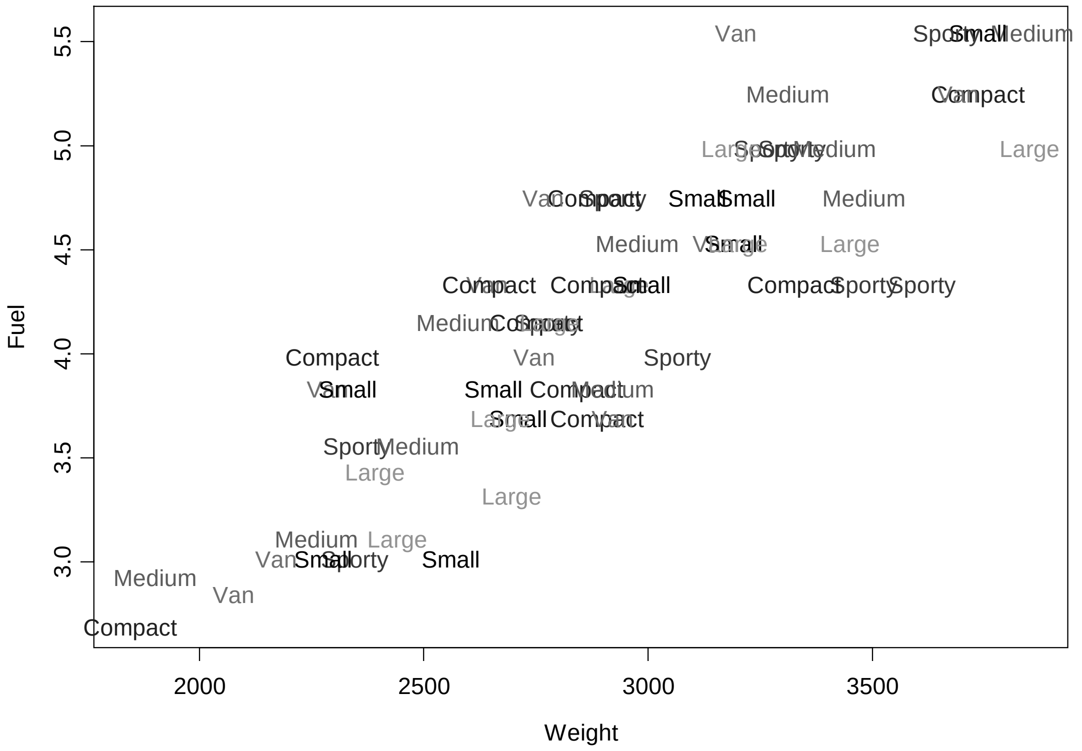
Example: Adding Points

```
> attach(fuel.frame)
> plot(Weight,Fuel,type="n",xlab="Weight \n Small = S, Medium = M")
> points(Weight[Type == "Small"],Fuel[Type == "Small"],pch="S")
> points(Weight[Type == "Medium"],Fuel[Type == "Medium"],pch="M")
```



Example: Adding Text

```
> size <- c("Small","Sporty","Compact","Medium","Large","Van")
> plot(Weight,Fuel,type="n")
> text(Weight,Fuel,labels=size,col=1:length(size))
```



Plot Types

- We can plot data in S-Plus in any of the following ways
  - as points
  - as lines
  - both lines and points(with points isolated)
  - as “overstruck” pints and lines (points not isolated)
  - as vertical line for each data point (known as “ high-density plot”)
  - as a stairstep plot
  - As an empty plot, with axes and labels but not data plotted.
- Different graphic functions have different default choices.
  - Scatter plot use points as defaults (plot)
  - Time series plot uses lines as the default (ts.plot)

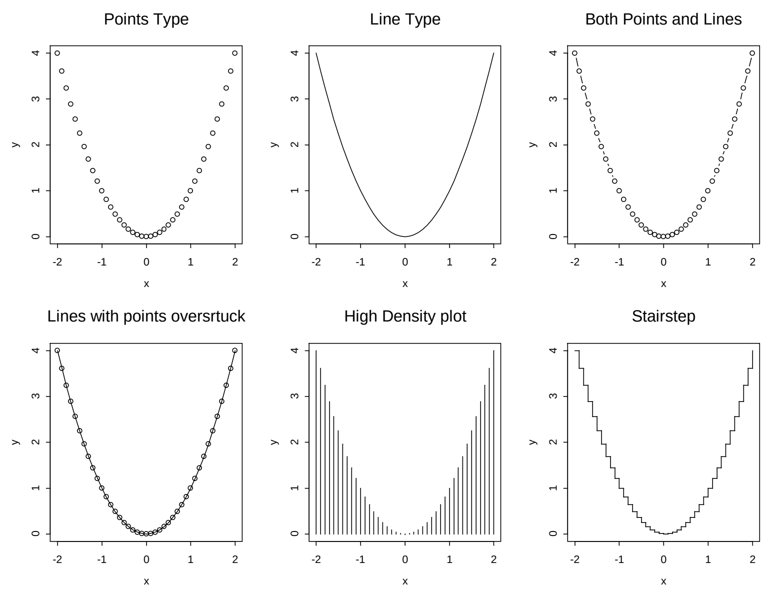
Choosing the Plot Type

- To choose the plot type we use type= .

– type="p"	points
– type="l"	lines
– type="b"	both points and lines
– type="o"	lines with points overstruck
– type="h"	high-density plot
– type="s"	stairstep plot
– type="n"	no data plotted

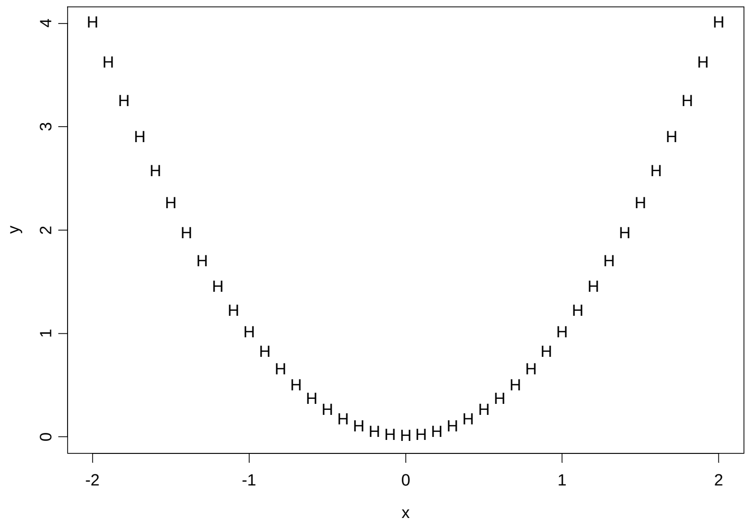
Example:

```
> par(mfrow=c(2,3))
> x <- seq(-2,2,0.1)
> y <- x^2
> plot(x,y,main="Points Type")
> plot(x,y,type="l",main="Line Type")
> plot(x,y,type="b",main="Both Points
    and Lines")
> plot(x,y,type="o",main="Lines with
    points oversrtuck")
> plot(x,y,type="h",main="High
    Density plot")
> plot(x,y,type="s",main="Stairstep")
```



Example:

```
> plot(x,y,type="p",pch="H")
```



Exercises: Graphic

Q1. a. Attach the **“ethanol”** data frame.

b. On the same graph sheet plot

  - histogram
  - qqnorm
  - boxplot

  for the variable NOx

c. Do a scatter plot of NOx vs E. Add a title. And use the plotting symbol to be “#”.

Q2. In a single figure, plot the functions sin, cos, sin+cos with different colors and line styles. Use 1000 points in the interval –2*pi and 2*pi and label the figure with a title, subtitle and axis label.

Q3. For the gun data create a boxplot for the variable Rounds for each of the Method. Do the same for Team.

Data Input and Output

Contents:

- Functions used to read data in S-PLUS
- Reading ASCII data into S-PLUS
- Reading Non-ASCII file format and databases into S-PLUS
- Exporting S-PLUS objects as
  - Text
  - Formatted and unformatted ASCII
  - Other file format and databases
- Exporting results from analysis and graphics from S-PLUS for report writing.

Functions used to read data in S-PLUS

- scan Function
- read.table Function
- import.data Function

scan Function

scan(file="", what=numeric(), n=<<see below>>,
    sep=<<see below>>, multi.line=F, flush=F,
    append=F, skip=0, widths=NULL,
    strip.white=<<see below>>)

OPTIONAL ARGUMENTS:

file a character string giving the name of the file to be scanned. If file is missing or empty (""), data will be read from standard input; scan will prompt with the index for the next data item, and data input can be terminated by a blank line. If file = "clipboard", scan reads from the Windows clipboard.

what a vector of mode numeric, character, or complex, or a list of vectors of these modes. Objects of mode logical are not allowed. If what is a numeric, character, or complex vector, scan will interpret all fields on the file as data of the same mode as that object. So, what=character() or what="" causes scan to read data as character fields. If what is missing, scan will interpret all fields as numeric.

If what is a list, then each record is considered to have length(what) fields and the mode of each field is the mode of the corresponding component in what. When widths is given as a vector of length greater than one, what must be a list of the same length as widths.

n	maximum number of items (number of records times fields per record) to read from the file. If omitted, the function reads to the end of file (or to an empty line, if reading from standard input).
sep	separator (single character), often "\t" for tab or "\n" for newline. If omitted, any amount of white space (blanks, tabs, and possibly newlines) can separate fields. If widths is specified, then sep tells what separator to insert into fixed-format records.
multi.line	if FALSE, all the fields must appear on one line: if scan reaches the end of the line without reading all the fields, an error occurs. Thus the number of fields on each line must be a multiple of the length of what unless flush=TRUE. This is useful for checking that no fields have been omitted. If this argument is TRUE, reading will continue, disregarding where new lines occur.

flush if TRUE, scan will flush to the end of the line after reading the last of the fields requested. This allows putting comments after the last field that are not read by scan, but also prevents putting multiple sets of items on one line.

append if TRUE, the returned object will include all the elements in the what argument, with the input data for the respective fields appended to each component. If FALSE (the default), the data in what is ignored, and only the modes matter.

skip the number of initial lines of the file that should be skipped prior to reading.

widths vector of integer field widths corresponding to items in the what argument. The widths argument provides for common fixed-format input. If widths is not NULL, then as scan reads the characters in a record, it automatically inserts a sep character after it reads widths[1] characters (widths[1] represents the width of the first field), then another sep after widths[2] characters, and so on, allowing the record to be read as if your input were delimited by the sep character to begin with. The default sep inserted when using widths is "1" (binary 1); if your input contains this character, you will need to set the sep argument to a character that you know is not contained anywhere in the input. One caveat: the widths you specify must correspond exactly to field widths in your input; if they do not, you may get "field undecipherable" errors in (seemingly) odd places, or the input may be silently but incorrectly digested. The default for widths is NULL. Note that if widths has a length greater than one, what must be a list of the same length.

strip.white vector of logical values corresponding to items in the what argument. The strip.white argument allows you to strip leading and trailing white space from character fields (scan always strips numeric fields in this way). If strip.white is not NULL, it must be either of length 1, in which case the single logical value tells whether to strip all fields read, or the same length as what, in which case the logical vector tells which fields to strip (strip the leading and trailing white space from field 1 if strip.white[1] is TRUE and field 1 is a character field, strip from field 2 if strip.white[2] is TRUE and field 2 is a character field, and so on). If widths is specified, the default for strip.white is TRUE (strip all fields), otherwise the default is NULL (do not strip any fields). Note: if you are reading free format input by leaving sep unspecified, then strip.white has no effect.

read.table Function

```
read.table(file, header=<<see below>>,
   sep, row.names, col.names,  as.is=F,
   na.strings="NA", skip=0)
```

REQUIRED ARGUMENTS:

file	character string naming the text file from which to read the data. The file should contain one line per row of the table. The fields may be separated by the character in sep, or the file may be fixed format with the fields starting at fixed points within each row. The value file = "clipboard" refers to the Windows clipboard.

OPTIONAL ARGUMENTS:

header logical flag: if TRUE, then the first line of the file is used as the variable names of the resulting data frame. The default is FALSE, unless there is one less field in the first line of the file than in the second line. Be careful, however, before accepting the default behavior. What looks like a valid header line may contain an empty cell, so that what appears to be a line with one less field than the other lines is actually a line with the same number of fields. In this case, header is set to FALSE. Such behavior may occur, for example, when copying a spreadsheet via the Clipboard. If you want to ensure that the headers and row names appear as in the Clipboard, use header=T and row.names=1.

sep the field separator (single character), often "\t" for tab. If omitted, any amount of white space (blanks or tabs) can separate fields. To read fixed format files, make sep a numeric vector giving the initial columns of the fields.

row.names optional specification of the row names for the data frame. If provided, it can give the actual row names, as a vector of length equal to the number of rows, or it can be a single number or character string. In the latter case, the argument indicates which variable in the data frame to use as row names (the variable will then be dropped from the frame). If row.names is missing, the function will use the first nonnumeric field with no duplicates as the row names. If no such field exists, the row names are 1:nrow(x). You can force this last version, regardless of suitable fields to use as row names, by giving row.names=NULL. Row names, wherever they come from, must be unique.

col.names	optional names for the variables. If missing, the header information, if any, is used; if all else fails, "V" and the field number are be pasted together. Variable names, wherever they come from, must be unique. Variable names will be converted to syntactic names before assignment, but not if they came from an explicit col.names argument.
as.is	control over conversions to factor objects. By default, non-numeric fields are turned into factors, except if they are used as row names. If some or all fields should be left as is (typically producing character variables), set the corresponding element of as.is to TRUE. The argument will be replicated as needed to be of length equal to the number of fields; thus, as.is=TRUE leaves all fields unconverted.

na.strings	character vector; when character data is converted to factor data the strings in na.strings will be excluded from the levels of the factor, so that if any of the character data were one of the strings in na.strings the corresponding element of the factor would be NA. Also, in a numeric column, these strings will be converted to NA.
skip	the number of lines in the file to skip before reading data.

import.data Function

```
import.data(DataFrame, FileName, FileType,
   TargetStartCol=1,EndCol="END",
   NameRow, StartRow, EndRow,ColNames,
   Format, Delimiters, Preview, Filter,
   OdbcConnection, OdbcSqlQuery,
   SeparateDelimeters=F)
```

REQUIRED ARGUMENTS:

DataFrame	a character string giving the name of the data frame to be created.
FileName	a character string giving the name of the file to import.
FileType	a character string specifying the type of file to import. It must be one of: "ACCESS", "ASCII", "DBASE", "EXCEL", "FASCII", "GAUSS", "LOTUS", "MATLAB", "ODBC", "PARADOX", "QUATTRO", "SAS", "SAS_TPT", "SPLUS", "SIGMAPLOT", "SPSS", "SPSS_POR", "STATA", "SYSTAT".

OPTIONAL ARGUMENTS:

TargetStartCol	an integer specifying the starting column in the source.
EndCol	an integer specifying the end column in the source, or the character string "END".
NameRow	the row containing the column names.
StartRow	an integer specifying the starting row from range in source.
EndRow	an integer specifying the end row from range in source.
ColNames	a vector of character strings to use as the column names in DataFrame.
Format	see notes on Importing ASCII Files, in the Importing and Exporting Data chapter in the User’s Guide.

Delimiters	range of characters that might be used as delimiters. Preview imports everything as characters.
Filter	see the chapter on Importing and Exporting Data in the Users Guide.
OdbcConnection	required if FileType="ODBC". Encrypted character string containing ODBC connection string.
OdbcSqlQuery	only meaningful if FileType="ODBC". It contains an optional SQL query. If no query is specified the first table of the data source is used.
SeparateDelimiters	a logical value; if TRUE, the separator is strictly a single character, else repeated consecutive separator characters are treated as one separator.

Reading ASCII data into S-PLUS

- Reading numeric data into a vector
- Reading non-numeric data into a vector
- Reading a Data Frame

Example1: Reading numeric data into a vector

- Import the file c:\datasets\score1.txt into S-PLUS and save it to the vector testscore1
  - First lets look at the contents of the file c:\datasets\score1.txt
  - Then do

```
> testscore1 <- scan("c:\\datasets\\score1.txt")
> testscore1
 [1]  9  8 10  7  8  9  1  0  9  9
```

Example2: Reading numeric data into a vector

- Import the file c:\datasets\score2.txt into S-PLUS and save it to the vector testscore1
  - First lets look at the contents of the file c:\datasets\score2.txt
  - Then do

```
> testscore2 <- scan("c:\\datasets\\score2.txt",
+                    sep=",")
> testscore2
[1]  6  5  5  7 17  5  6  5
```

Example3: Reading non-numeric data into a vector

- Import the file c:\datasets\names1.txt into S-PLUS and save it to the vector class.names
  - First lets look at the contents of the file c:\datasets\names1.txt
  - Then do

```
> names <- scan("c:\\datasets\\names1.txt",what="")
> names
 [1] "Andreas" "Sharon"  "Kevin"   "Sam"
 [5] "David"   "Alice"   "Cindy"   "Dan"
 [9] "Mary"    "Ken"
```

Example4: Reading a Data Frame

- Import the file c:\datasets\authors.txt into S-PLUS and save it to the data frame authors.df
  - First lets look at the contents of the file c:\datasets\authors.txt
  - Then do

```
> authors.df <-
  read.table("c:\\datasets\\authors.txt")
```

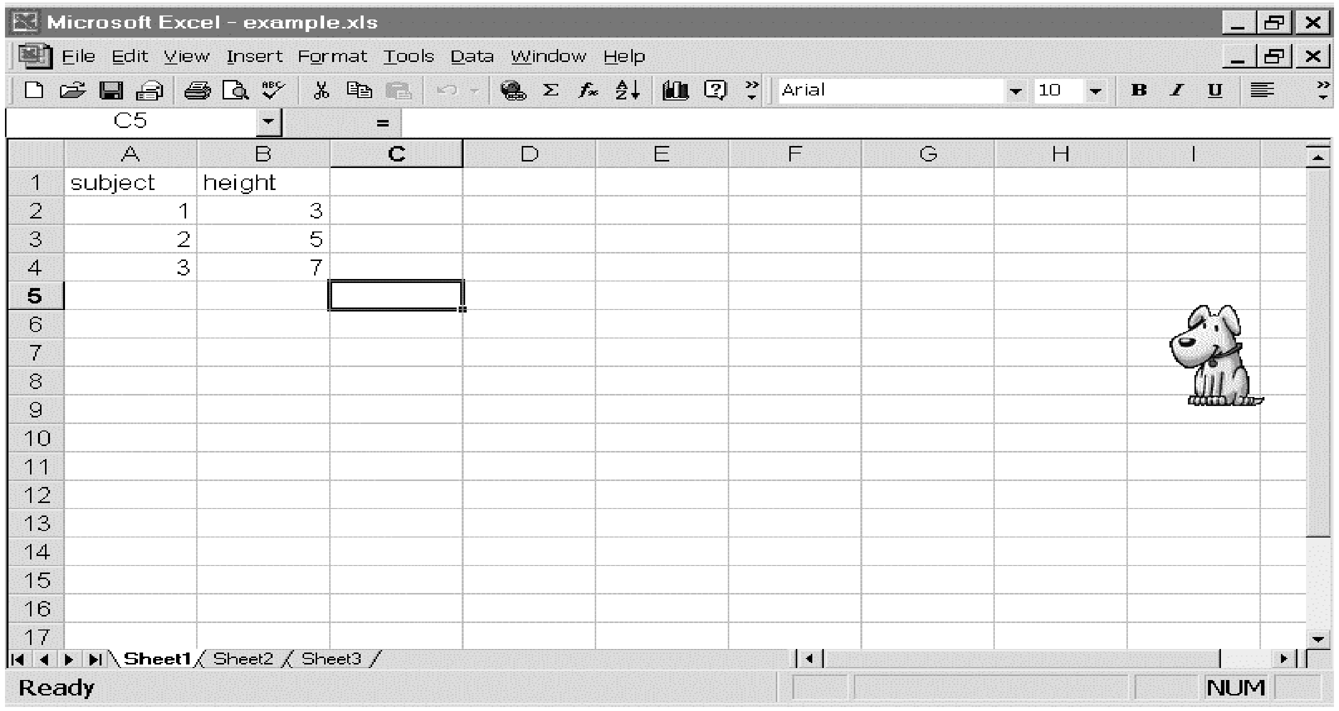
```
> authors.df
  FirstName LastName Age  Income       Home
1     Lorne    Green  82 1200000 California
2     Loren     Jaye  40   40000 Washington
3     Robin    Green  45   25000 Washington
4     Robin     Howe   2       0    Alberta
5     Billy     Jaye  40   27500 Washington
```

Reading Non-ASCII data into S-PLUS

- Using read.table where file="clipboard"
  - It is an easy way to import formatted text into SPLUS by pasting formatted text to the clipboard and then reading it into S-Plus using read.table where file="clipboard"
- Using import.data

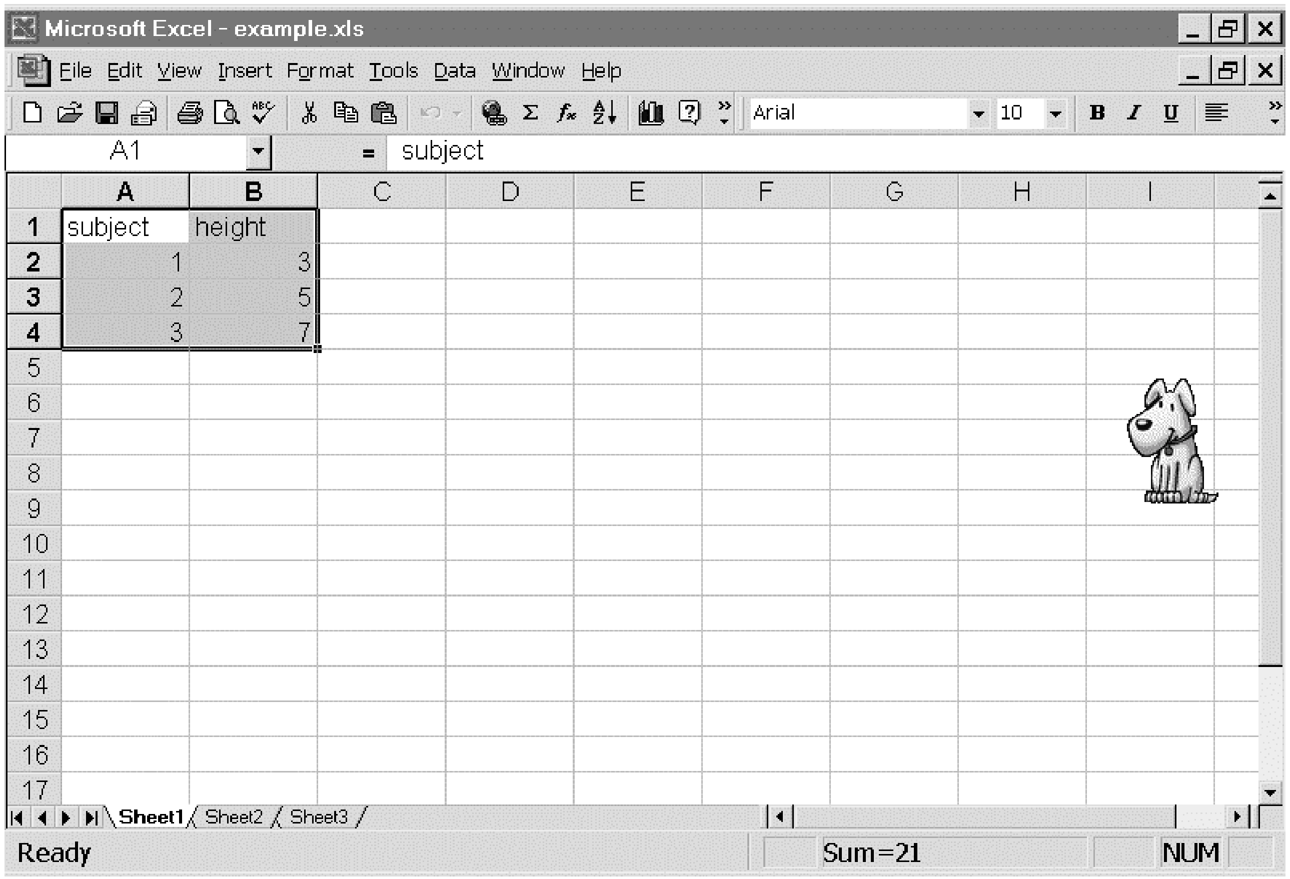
Example 5: Using read.table where file="clipboard"

- Suppose you have the file c:\datasets\example.xls. We want to import the data into SPLUS


Microsoft Excel - example.xls
File Edit View Insert Format Tools Data Window Help
Arial
10
C5
Sheet1
Sheet2
Sheet3
Ready
NUM

	A	B
1	subject	height
2	1	3
3	2	5
4	3	7

- Highlight the area you want to import


Microsoft Excel - example.xls
File Edit View Insert Format Tools Data Window Help
Arial
10
A1
= subject
Sheet1 Sheet2 Sheet3
Ready
Sum=21
NUM

	A	B
1	subject	height
2	1	3
3	2	5
4	3	7

- Copy to the clipboard by CTRL-C

- In SPLUS do the following

```
> read.table(file="clipboard",header=T)
  subject height
1       1      3
2       2      5
3       3      7
```

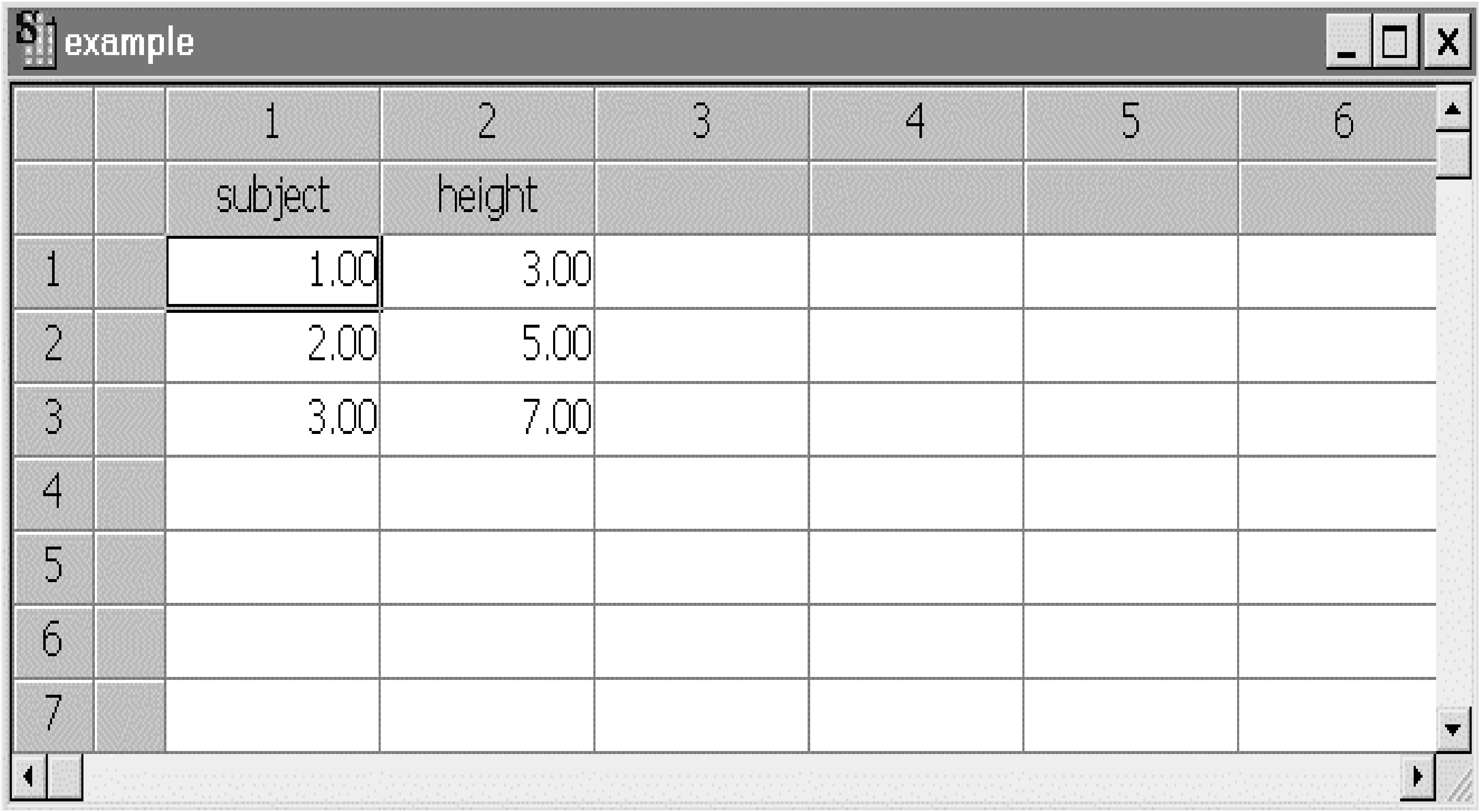
- Note: **header=T** was used because the first raw of the file is the variable name

Example 6: Using import.data function

- Import the file example.xls using the function import.data
- To do so you type the following in S-PLUS

```
> import.data("c:\\datasets\\example.xls",
+   FileType="EXCEL",
+   NameRow="1",
+   DataFrame="example")
```

- The result is



		1	2	3	4	5	6
		subject	height				
1		1.00	3.00				
2		2.00	5.00				
3		3.00	7.00				
4							
5							
6							
7							

Functions used to read data out from S-PLUS

- export.data
- data.dump function and data.restore
- sink
- write.table

export.data Function

- export.data(DataSet, FileName, FileType, ...)

REQUIRED ARGUMENTS:

DataSet	a character string giving the name of the data object of class data.frame, matrix or vector to be created.
FileName	a character string giving the name of the file and directory
FileType	one of "ACCESS", "ASCII", "DBASE", "EXCEL", "FASCII", "GAUSS", "LOTUS", "MATLAB", "ODBC", "PARADOX", "QUATTRO", "SAS", "SAS_TPT", "SPLUS", "SIGMAPLOT", "SPSS", "SPSS_POR", "STATA", "SYSTAT".

OPTIONAL ARGUMENTS:

Columns	specify a subset of the columns to be exported.
Rows	specify a subset of the rows to be exported.
Delimiter	character to be used as delimiter.
ColumnNames	output names: true or false
Quotes	quotes around characters: true or false
LineLength	maximum length of one line

OdbcConnection required and meaningful only if FileType = "ODBC". This is an encrypted character string containing the ODBC connection string.

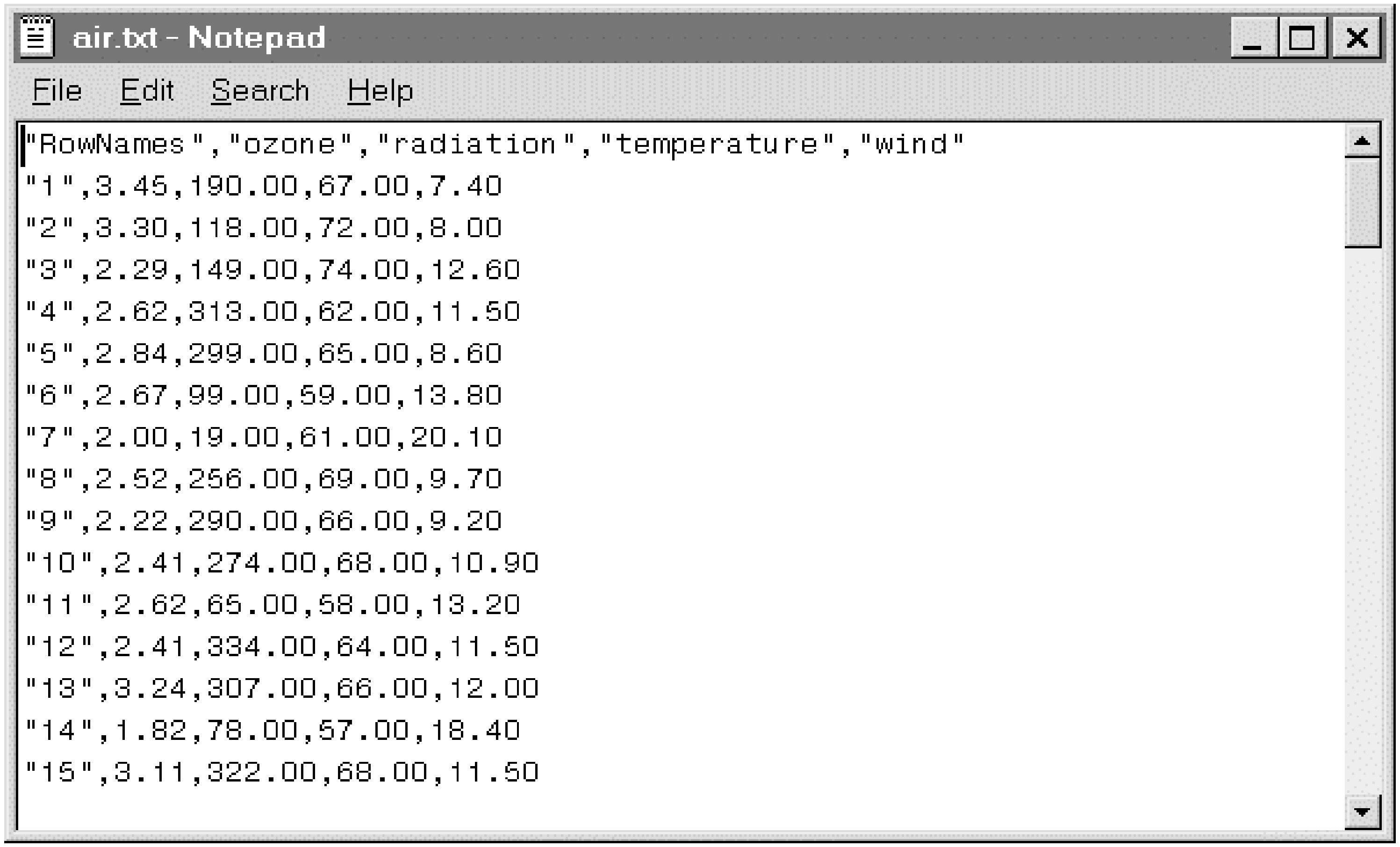
OdbcTable required name of table to be created if FileType ="ODBC".

Example 7: Export data frame into an ASCII file Format

Export the data frame air into the file
c:\datasets\air.txt
In S-PLUS do the following:

```
> export.data(DataSet="air",
        Delimiter=",",
        ColumnNames=T,
        Quotes=T,
        FileType="ASCII",
        FileName="c:\\datasets\\air.txt")
```

- The file “**air.txt**” is created in the directory “c:\datasets”. It looks as follows



```
"RowNames","ozone","radiation","temperature","wind"
"1",3.45,190.00,67.00,7.40
"2",3.30,118.00,72.00,8.00
"3",2.29,149.00,74.00,12.60
"4",2.62,313.00,62.00,11.50
"5",2.84,299.00,65.00,8.60
"6",2.67,99.00,59.00,13.80
"7",2.00,19.00,61.00,20.10
"8",2.52,256.00,69.00,9.70
"9",2.22,290.00,66.00,9.20
"10",2.41,274.00,68.00,10.90
"11",2.62,65.00,58.00,13.20
"12",2.41,334.00,64.00,11.50
"13",3.24,307.00,66.00,12.00
"14",1.82,78.00,57.00,18.40
"15",3.11,322.00,68.00,11.50
```

data.dump & data.restore

- data.dump(names,file=“dumpdata”)

- It transfer an S-Plus data object and functions to another person
- It is considered to be the most efficient way for exporting S-PLUS object
- You can export more than one object to the same file
- You have to use data.restore() to recreate the object in another session.

Example 8: Export two data frames Using data.dump()

```
> data.dump(c("air","fuel.frame"),
+ file="c:\\datasets\\data.dmp")
[1] "c:\\datasets\\data.dmp"
```

In another S-PLUS session you can restore the data by

```
> data.restore(“c:\datasets\\data.dump”)
```

Sink function

- sink(file="filename")
- It is used to save the output for reading later by redirecting the output to a file.
- To do that you do the following

```
>sink("filename")
S-PLUS stuff…
>sink()
```

- To echo the input to a file along with the output use the set **options(echo=T)**

Example 9: sink() function

```
> sink("c:\\datasets\\ouput.txt")
> x <- 1:10
> x
> y <- x/20
> y
> sink()
```

The ouput.txt file will contain the following:

```
 [1]  1  2  3  4  5  6  7  8  9 10
 [1] 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45
   0.50
```

Example 9: sink() function using options(echo=T)

```
> options(echo=T)
options(echo = T)
> sink("c:\\datasets\\output.txt")
sink("c:\\datasets\\output.txt")
> x <- 1:5
> y <- x*2
> x
> y
> sink()
```

The file output.txt contains:

```
x <- 1:5
y <- x * 2
x
[1] 1 2 3 4 5
y
[1]  2  4  6  8 10
sink()
```

write.table function

- write.table(data, file = "", sep = ",", append = F, quote.strings = F, dimnames.write = T, na = NA, end.of.row = "\n")
- It can be used to save matrices and data frame as ASCII.

Example 10: write.table() function

```
> write.table(fuel.frame,"c:\\datasets\\fuel.txt")
write.table(fuel.frame, "c:\\datasets\\fuel.txt")
```

write function

- write(x, file="data", ncolumns=<<see below>>, append=F)

- It writes the contents of the data to a file in ASCII format.

Example 11: export.graph() function

```
> write(x,"c:\\datasets\\x.txt")
```

- The file x.txt contains

```
1 2 3 4 5
```

Exporting Graphs

- export.graph function
- Syntax

export.graph(FileName, Name,
ExportType=<<see below>>, ...)

REQUIRED ARGUMENTS

• FileName	a character string giving the name of the file to be created.
• Name	a character string specifying the object path name for a graphsheet.

OPTIONAL ARGUMENTS

- ExportType one of the character strings: "BMP", "CGM", "EPS TIFF" (EPS w/TIFF), "EPS", "EPS PRINT" (EPS using PostScript printer driver), "GIF", "HGL", "IMG", "JPG", "MET", "PCL", "X", "PICT", "SDW" (AmiPro), "TIFF", "TGA", "WMF", "WPB" (WordPerfect Bitmap), "WPV" (WordPerfect Vector). If ExportType is not specified, the file type will be inferred from the file extension used in the FileName argument.

Example 12: export.graph() function

```
> attach(air)
> plot(wind,ozone)
>export.graph(Name="GSD2",ExportType="
   GIF“,
   FileName="c:\\datasets\\mygraph.gif")
```

- A file name mygraph.gif is created.

Exercises: Data Import & Export

Q1. Import the file c:\datasets\Gpadata.sd2 into SPLUS. (Note that Gpadata.sd2 is a SAS file)?

Q2. Import the file c:\datasets\gpa.txt into SPLUS?

Q3. Export the data frame gas data frame to c:\datasets\gas.txt?

Q4. Export the data frame gas data frame to c:\datasets\gas.sd2?

Q5. Use the function data.dump() to transfer the gun and car.all data frame to another S-PLUS session.

Q6. Write the data frame galaxy to a text file using the sink() function?

Q7. Create a graph and export it to c:\datasets\graph.gif?

Functions

Contents:

- Functions and Operators
- Writing Functions

Functions and Operators

- Calling an S-PLUS Function

Calling an S-PLUS Function

- All functions in S-Plus have both required and optional arguments
  - Arguments are enclosed in parentheses and separated by commas.
  - The help file explains which arguments are required and which are optional.
  - The **args** function is used to find out the arguments of the function
  - Arguments with “=“ sign have a default values and are optional.

Example:

- What are the argument of the sum function?

```
> args(mean)
function(x, trim = 0, na.rm = F)
NULL
```

So,

REQUIRED ARGUMENTS

x	numeric object. Missing values (NAs) are allowed.

OPTIONAL ARGUMENTS

trim	fraction (between 0 and .5, inclusive) of values to be trimmed from each end of the ordered data. If trim=.5, the result is the median.
na.rm	logical flag: should missing values be removed before computation?

- To invoke a function you must supply parentheses even if no argument is required. Otherwise it will print out the definition of the function

Example:

```
> search
function()
.Internal(search(), "S_database", T, 3)
> search()
 [1] "d:\\sp2000\\users\\fhilal\\_Data"
 [2] "d:\\sp2000\\splus\\_Functio"
```

- Arguments must be supplied by position, by name or both
  - If they are supplied by position, then they should be entered exactly in the same order as in the function definition

Example:

```
> y
[1] 1.0 1.1 1.3 2.0 2.4
> mean(y)
[1] 1.56
> # Computes the 20% trimmed mean
> mean(y,0.2)
[1] 1.466667
```

- If they are supplied by name, then they may entered in any order. The name should be followed by the equal sign and then the argument value.
- Note that the argument name can be abbreviated to the first few characters as long as the abbreviation uniquely identifies the argument

Examples:

```
> mean(x=y,trim=0.2)

[1] 1.466667

> mean(trim=0.2,x=y)

[1] 1.466667

> mean(tr=0.2,x=y)

[1] 1.466667
```

Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponential
-	Unary minus
%/%	Integer Divide
%%	Module
%*%	Matrix multiplication
%o%	Outer product
:	Sequence

•All operators except the urinary Minus need two operands, one to the left and one to the right.

•The operands may be of single numbers, vectors, or matrices, as long as the operation makes sense.

Examples:

```
> 8+10
[1] 18
> 8-10
[1] -2
> 8*10
[1] 80
> 8/10
[1] 0.8
> 8^10
[1] 1073741824
```

```
> -8+10
[1] 2
> 10 %/% 8
[1] 1
> 10 %% 8
[1] 2
```

```
> mat1 <- matrix(1:6,nrow=2)
> mat1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> mat2 <- matrix(1:3,nrow=3)
> mat1%*%mat2
     [,1]
[1,]   22
[2,]   28
```

Examples:

```
> (1:3) %o% (4:5)
     [,1] [,2]
[1,]    4    5
[2,]    8   10
[3,]   12   15
> 4:9
[1] 4 5 6 7 8 9
```

Vector Arithmetic

- It is performed element-by-element
- If the length of one element is less than the other then the shorter one is cyclically replicated to the same length as the longer one.

Examples:

```
> vec1 <- c(2,6,9)
> vec1
[1] 2 6 9
> vec2 <- vec1^2
> vec2
[1]  4 36 81
> vec1+vec2
[1]  6 42 90
```

```
> mat1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> mat1*mat1
     [,1] [,2] [,3]
[1,]    1    9   25
[2,]    4   16   36
```

Examples:

```
> vec1-3
[1] -1  3  6
> mat1-4
     [,1] [,2] [,3]
[1,]   -3   -1    1
[2,]   -2    0    2
> mat1-c(1,2)
     [,1] [,2] [,3]
[1,]    0    2    4
[2,]    0    2    4
```

```
> vec1+(3:8)
[1]  5 10 14  8 13 17
> vec1+(3:7)
[1]  5 10 14  8 13
Warning messages:
  Length of longer object is
not a multiple of the length of
the shorter object in: ve
c1 + (3:7)
```

Logical Operators

Operator	Description
==	Equals
!=	Not Equals
<	Less Than
<=	Less Than or Equal
>	Greater Than
>=	Greater Than or Equal
&	Elementwise And
|	Elementwise Or
&&	Control And
||	Control Or
!	Not

- All operators except the Urinary Not need two operands, one to the left and one to the right.
- The operands may be of single numbers, vectors, or matrices, as long as the operation makes sense.

Examples:

```
> vec1
[1] 2 6 9
> vec1 == 9
[1] F F T
> vec1 != 9
[1] T T F
> vec1 < 4
[1] T F F
> vec1 <= 6
[1] T T F
```

```
> vec1 > 6
[1] F F T
> vec1 >= 6
[1] F T T
```

Examples:

```
> vec1
[1] 2 6 9
> vec2
[1]  4 36 81
> vec1 < 7 & vec2 <=10
[1] T F F
> (!(vec1 < 7 & vec2 <=10))
[1] F T T
> vec1 < 7 | vec2 <=10
[1] T T F
```

```
> mode(vec1)!="character" &&
 min(vec1)>0
[1] T
> mode(vec1)!="character" ||
min(vec1)>0
[1] T
```

•You can use a combination of logical operators and subscripting to extract data .

Example:

```
> vec3 <- 1:20

> vec4 <- vec3[vec3 < 7]

> vec4

[1] 1 2 3 4 5 6
```

•Four functions that are related to logical operators

- **any**. Logical sum. It tests to see if any elements are TRUE.
- **all**. Logical product. It tests to see if all elements are TRUE.
- **all.equal**. Test whether two objects are identical
- **objdiff**. Displays differences between two objects.

Examples:

```
> all(vec1>7)
[1] F
> any(vec1>7)
[1] T
```

Operator Precedence in S-PLUS

Operator	**Explanation**
{	Expression delimiter (Curly Brace)
(	Expression delimiter (parenthesis)
$	List and data frame extraction
[[[	Subscripts
^	Exponentiation
-	Urinary minus
:	Sequence

Operator	**Explanation**
%*% %/% %%	Matrix multiplication, Integer Division, Modulo
* /	Multiplication and division
+ -	Addition and subtraction
< > <= >= == !=	Logical comparisons
!	Unary not
& && | ||	Logical operators
<<-	Global assignment
<- _ ->	Assignments

Writing Functions

- There are two types of functions
  - Built in functions
  - User-created function
- Writing functions can be done in two way:
  - Modify an existing function
  - Write original function

Built-in Functions

To call a Built-in S-Plus function:

at the command line type

> *name*(*arguments*)

where,

- *name*: is the name of the function
- *arguments*: argument are required without a specified default value.

Examples of Built-in-Functions

- mean(x): gives the arithmetic mean of x.
- var(x): gives the variance of x.
- stem(x): draw a stem-and-leaf plot for x.
- floor(x): next smaller integer.
- sum(x): gives the sum of x.
- log10(x): gives the base 10 logarithm of x.
- cor(x, y): gives the correlation between x and y
- plot(x, y): plot x versus y.

Examples:

```
> attach(air)

> mean(ozone)

[1] 3.247784

> cor(ozone,wind)

[1] -0.5989278

> var(wind)

[1] 12.66803
```

```
> stem(wind)

N = 111   Median = 9.7
Quartiles = 7.4, 11.5
Decimal point is at the colon

   2 : 38
   3 : 4
   4 : 0066
   5 : 11177
   6 : 333333999999
   7 : 444444444
   8 : 0000000666
   9 : 222222777777777
  10 : 3333333333999999
  11 : 5555555555
  12 : 000066
  13 : 28888
  14 : 33339999
  15 : 555
  16 : 66
  17 :
  18 : 4
High: 20.1 20.7
```

General Syntax of a Function

Function(arguments) {

  body

  }

Where,

•***arguments*** give the arguments of the function separated by commas

•***body*** is the body of the function made up of one or more S-PLUS expressions

- Any S-PLUS objects can be passed as an argument including functions.
- Variables defined within the body of the function are local to that function. Which mean that only last as long as the function is executing.
- You can make a global assignment within the body of the function by using the global assignment <<- or the function **assign**.
- A function returns the value of the last evaluated expression in the body of the function.

- To create or modify functions within S-PLUS do one of the following
  - At the command line, use an assignment statement followed by the function definition
  - Use the ***edit*** (ed if UNIX) or the ***fix*** functions to use an editor within S-PLUS.
- To create or modify a function outside S-PLUS do the following
  - For Unix use the UNIX editor like vi or emacs and for windows use the Notepad to create an ASCII file with a function definition, then use the ***source*** function to read the file in S-PLUS.

Example:

Create a function to calculate the standard deviation.

Method I:

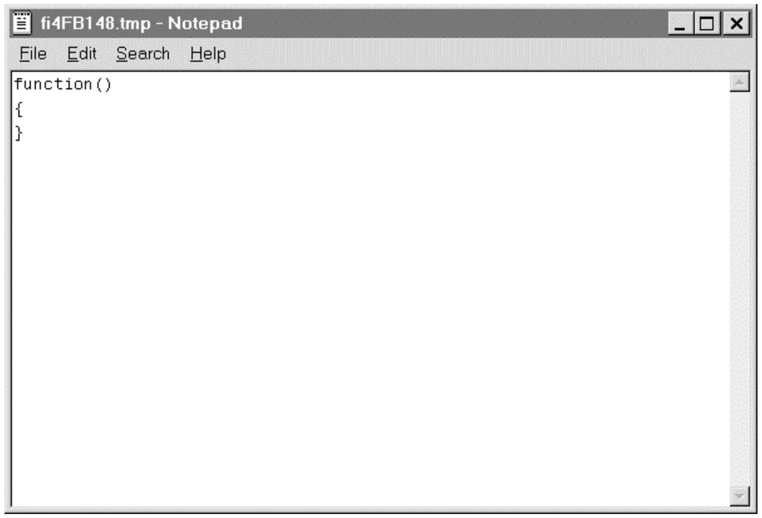
```
> st.dev <- function (x) {sqrt(var(x))}
> st.dev(vec1)
[1] 3.511885
```

Method II:

```
> fix(st.dev)
```

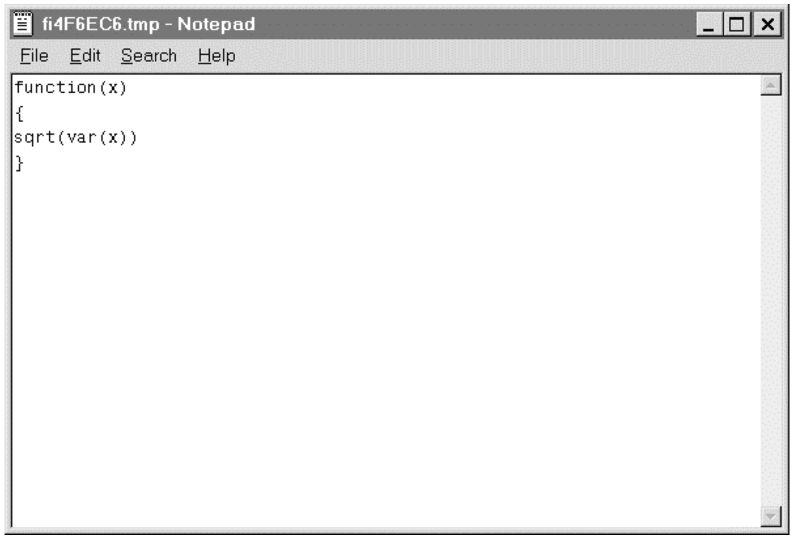
This will open the following window:

```
> st.dev(vec2)
[1] 38.68247
```



```
function()
{
}
```

Then fill the appropriate code as follows:



```
function(x)
{
sqrt(var(x))
}
```

Then save and exit.

Test the function:

Arguments

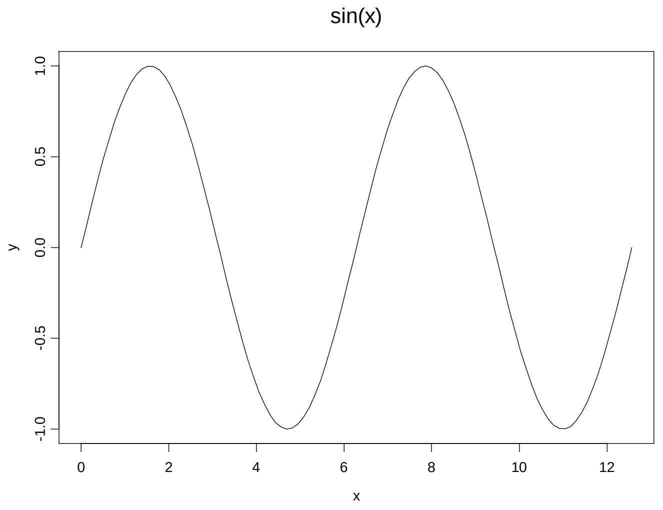
- There are three basic types of arguments
  - Required arguments
  - Optional arguments with default values
  - Variables arguments
- Any object may be passed as an argument
- Arguments do not have to be explicitly typed (it might accept either a character vector or a logical matrix)

- Required argument must be listed before default arguments.
- Arguments are evaluated only as needed when they appear in the body of the function.
- Default arguments are specified by following the argument name by an equal sign and an expression that evaluates to the default value.

Example: ***Passing variable arguments***

Create a function to plot the sin function evaluated at numx values between minx and maxx.

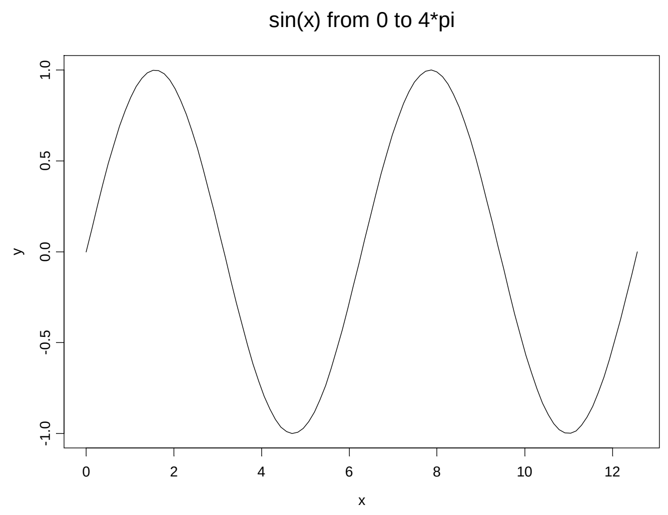
```
> sin.plot <- function(minx,maxx,numx=100,type="l"){
+     x <- seq(minx,maxx,length=100)
+     y <- sin(x)
+     plot(x,y,type=type)
+     }
> sin.plot(0,4*pi)
> title("sin(x)")
```



Example:

Create a function to plot the sin function evaluated at numx values between minx and maxx.

```
> sin.plot <- function(minx,maxx,numx=100,type="l“,…){
+     x <- seq(minx,maxx,length=100)
+     y <- sin(x)
+     plot(x,y,type=type,…)
+     }
```



```
> sin.plot(0,4*pi,main=“sin(x) from 0 to 4*pi”)
```

Conditionals

- To perform conditional evaluation of expression we need to use
  - if
  - if-else
  - ifelse function
- The if statement is used to evaluate an expression if some condition is true. And an else statement is used if an alternate action is needed
- The ifelse function takes a logical vector along with vectors of values to return if each element of the vector is true or false.

Syntax:

- if (condition) {expression}
- if (condition) {

  expression1}

  else {

      expression2

      }
- ifelse(test.vec,yes.vec,no.vec)

Example:

•Avoid getting an error message when x is a character of <= 0.

```
> x <-3
>   if(mode(x)!="character" && min(x)>0) log(x)
[1] 1.098612
> x <- 3
>   if(mode(x)!="character" && min(x)>0) log(x)
[1] 1.098612
> x <- "a"
>   if(mode(x)!="character" && min(x)>0) log(x)
NULL
```

Example:

- Give a message saying that you have invalid data

```
> myfun <- function(x){
+   if(mode(x)!="character" && min(x)>0) out<-log(x)
+   else out <- "not valid value"
+   out
+  }
> y <- 4
> myfun(y)
[1] 1.386294
> y <- "b"
> myfun(y)
[1] "not valid value"
```

Example:

•Function that will return the smaller of the two numbers:

```
> smaller <- function(x,y)
+ {}
> smaller <- function(x,y){
+    ifelse(x<y,x,y)
+ }
> smaller(1:5,5:1)
[1] 1 2 3 2 1
```

Iteration

- Use **for** to loop over the values of a vector
  - for (*name* in *value*) {*expression*}
- Use **while** to repeat until some condition changes
  - while (*condition*) {*expression*}
- Use **apply** to repeat a procedure for all rows or columns of a matrix
  - apply(*data*, *margin*, *function*,..)

Example:

```
> for(i in 1:5)
+ { print(i)}
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

```
> i <- 0
> while(i < 4){
+    print("i is less than 4")
+    i <- i+1
+    }
[1] "i is less than 4"
[1] "i is less than 4"
[1] "i is less than 4"
[1] "i is less than 4"
```

Example:

```
> mat1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> apply(mat1,2,mean)
[1] 1.5 3.5 5.5
> apply(mat1,1,mean)
[1] 3 4
```

Exercises: Functions

Q1. Create a function that will take one vector of data x as an input and will create a histogram. Try the function.

Q2. Modify the above function to create on the same graphic window a

- histogram
- boxplot

Try the function

Q3. Create a function that will take a vectors x and it will plot the square root of x. If the data is invalid then to print a message saying. Try the function. You need to use two built in functions the **any** function and the **warning** function.

Q4. Create a function that plots any function evaluated at a certain number of values of x between min.x and max.x. On the ylab specify the name of the function you plotted also give the plot a title. I want the ylab and the title to be arguments of the function.

Q5. Create a function that will find the coefficient of variation of a set of data.

The coefficient of variation is the ration between the standard deviation and the mean.

Q6. The above function will not work if there is missing values in the data. Modify the function you created in Q5 to discard the missing values. You need to use the function **is.na**.

Solutions

Solution: Introduction

Q1. search()

Q2. find(glm)

    or, find(glm,numeric=T)

Q3. objects(2,pattern="*glm*")

Solution:Data Objects

Q1.

```
> Letters <- c(T,F)
> mode(Letters)
[1] "logical"
> Letters <- c("T","F")
> mode(Letters)
[1] "character“
```

Q2. I get the following:

```
Error in num1 + num2: Non-
numeric first operand
```

Q3.

```
> mode(num1)<- "numeric"
> mode(num2)<- "numeric"
> num1+num2
[1] 5
```

Q4. a.

```
> rep(5,1)
[1] 5
> rep(1,5)
[1] 1 1 1 1 1
> rep(c(0,6),2)
[1] 0 6 0 6
> rep(c("a","b"),3)
[1] "a" "b" "a" "b" "a" "b"
> rep(1:3,4)
 [1] 1 2 3 1 2 3 1 2 3 1 2 3
> rep(c(1,5,8),length=10)
 [1] 1 5 8 1 5 8 1 5 8 1
> rep(1:5,1:5)
 [1] 1 2 2 3 3 3 4 4 4 4 5 5 5 5
    5
```

Q4. b.

```
> x <- rep(1:5,1:5)
> length(x)
[1] 15
```

Q4.c.

```
> length(x) <- 5
> x
[1] 1 2 2 3 3
```

So, you truncated x to only 5 elements.

Q5.

```
> seq(1,5,1)
[1] 1 2 3 4 5
> seq(1,5)
[1] 1 2 3 4 5
> seq(5)
[1] 1 2 3 4 5
> seq(5,1,-1)
[1] 5 4 3 2 1
> seq(5:1)
[1] 1 2 3 4 5
> 5:1
[1] 5 4 3 2 1
> seq(3,4,0.1)
 [1] 3.0 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 4.0
```

Q6.

```
> x <- seq(1,15,1)
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
> y <- x>5
> y
 [1] F F F F F T T T T T T T T T T
> z <- x[x>5]
> z
 [1]  6  7  8  9 10 11 12 13 14 15
```

Q7.

```
> grade <- c(5,7,8)
> name <- c("Sam","Dan","Susan")
```

a.

```
> c(grade,name)
[1] "5"     "7"     "8"     "Sam"   "Dan"   "Susan“
```

So the numerical values where forced into characters.

b.

```
> c(grade,T,F,T,F)
[1] 5 7 8 1 0 1 0
```

So, the logical values where forced into numerical values.

Q8.

```
> mat <- matrix(1:6,ncol=3)
> mat
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> mat <- matrix(1:6,ncol=3,byrow=T)
> mat
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Q9.a.

```
> grade1 <- c(5,7,8)
> grade2 <- c(6,6,10)
> grade.mat <- rbind(grade1,grade2)
> grade.mat
       [,1] [,2] [,3]
grade1    5    7    8
grade2    6    6   10
```

Q9.b.

```
> mode(grade.mat)
[1] "numeric"
```

Q9.c.

```
> dim(grade.mat)
[1] 2 3
```

Q9.d.

```
> dimnames(grade.mat)
[[1]]:
[1] "grade1" "grade2"

[[2]]:
character(0)
```

Q9.e.

```
> name <- c("Sam","Dan","Suzan")
> dimnames(grade.mat)[[2]] <- name
> grade.mat
       Sam Dan Suzan
grade1   5   7     8
grade2   6   6    10
```

Q9.f.

```
> grade.mat[,1]
 grade1 grade2
      5      6
> grade.mat[,"Sam"]
 grade1 grade2
      5      6
```

Q9.g.

```
> grade.mat[,c(T,F,T)]
       Sam Suzan
grade1   5     8
grade2   6    10
```

Q9.h.

```
> t(grade.mat)
      grade1 grade2
  Sam      5      6
  Dan      7      6
Suzan      8     10
```

Q10.a.

```
> FirstName <- c("Lorne","Loren","Robin","Robin","Billy")
> LastName <- c("Green","Jaye","Green","Howe","Jaye")
> Age <- c(82,40,45,2,40)
> Income <- c(1200000,40000,25000,0,27500)
> Home <-
    c("California","Washington","Washington","Alberta","Washington")
> authors <- data.frame(FirstName,LastName,Age,Income,Home)
> authors
  FirstName LastName Age  Income       Home
1     Lorne    Green  82 1200000 California
2     Loren     Jaye  40   40000 Washington
3     Robin    Green  45   25000 Washington
4     Robin     Howe   2       0    Alberta
5     Billy     Jaye  40   27500 Washington
```

Q10.b.

```
> authors$Age[1:3]
[1] 82 40 45
> authors[1:3]$Age
[1] 82 40 45  2 40
```

Q11.a.

```
> class(air)
[1] "data.frame“
```

Q11.b.

```
> names(air)
[1] "ozone"       "radiation"   "temperature" "wind"
```

Q11.c.

```
> row.names(air)
  [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "11"  "12"  "13"
 [14] "14"  "15"  "16"  "17"  "18"  "19"  "20"  "21"  "22"  "23"  "24"  "25"  "26"
 [27] "27"  "28"  "29"  "30"  "31"  "32"  "33"  "34"  "35"  "36"  "37"  "38"  "39"
 [40] "40"  "41"  "42"  "43"  "44"  "45"  "46"  "47"  "48"  "49"  "50"  "51"  "52"
 [53] "53"  "54"  "55"  "56"  "57"  "58"  "59"  "60"  "61"  "62"  "63"  "64"  "65"
 [66] "66"  "67"  "68"  "69"  "70"  "71"  "72"  "73"  "74"  "75"  "76"  "77"  "78"
 [79] "79"  "80"  "81"  "82"  "83"  "84"  "85"  "86"  "87"  "88"  "89"  "90"  "91"
 [92] "92"  "93"  "94"  "95"  "96"  "97"  "98"  "99"  "100" "101" "102" "103" "104"
[105] "105" "106" "107" "108" "109" "110" "111"
```

Q12.a.

```
> list1 <- list(a=1:3,b=rep(4,5),l=letters[1:5])
> list2 <- list(list1=list1,old=list(1:5,7:4),c(2,3))
```

Q12.b.

```
> names(list1)
[1] "a" "b" "l"
> names(list2)
[1] "list1" "old"   ""
```

Q12.c.

```
> list1$b
[1] 4 4 4 4 4
> list2$list1$l
[1] "a" "b" "c" "d" "e"
> list2[[3]]
[1] 2 3
> list2[[3]]+4
[1] 6 7
> list2[3]+4
Error in list2[3] + 4: Non-numeric first operand
```

Q13.a.

```
> age.factor <- factor(cut(age,
+  breaks=c(0,30,60,90)),
+  levels=c(1,2,3),
+  label=c("Young","Middle Age","Old"))
> age.factor
[1] Young      Old        Young      Middle Age Middle Age Young      Old
[8] Old
```

Q13.a.

```
> attributes(age.factor)
$levels:
[1] "Young"      "Middle Age" "Old"

$class:
[1] "factor"
```

Solutions:Statistical Models

Q1.a.

```
> swiss.df <- data.frame(swiss.fertility,swiss.x)
```

Q1.b.

```
> summary(swiss.df)
```

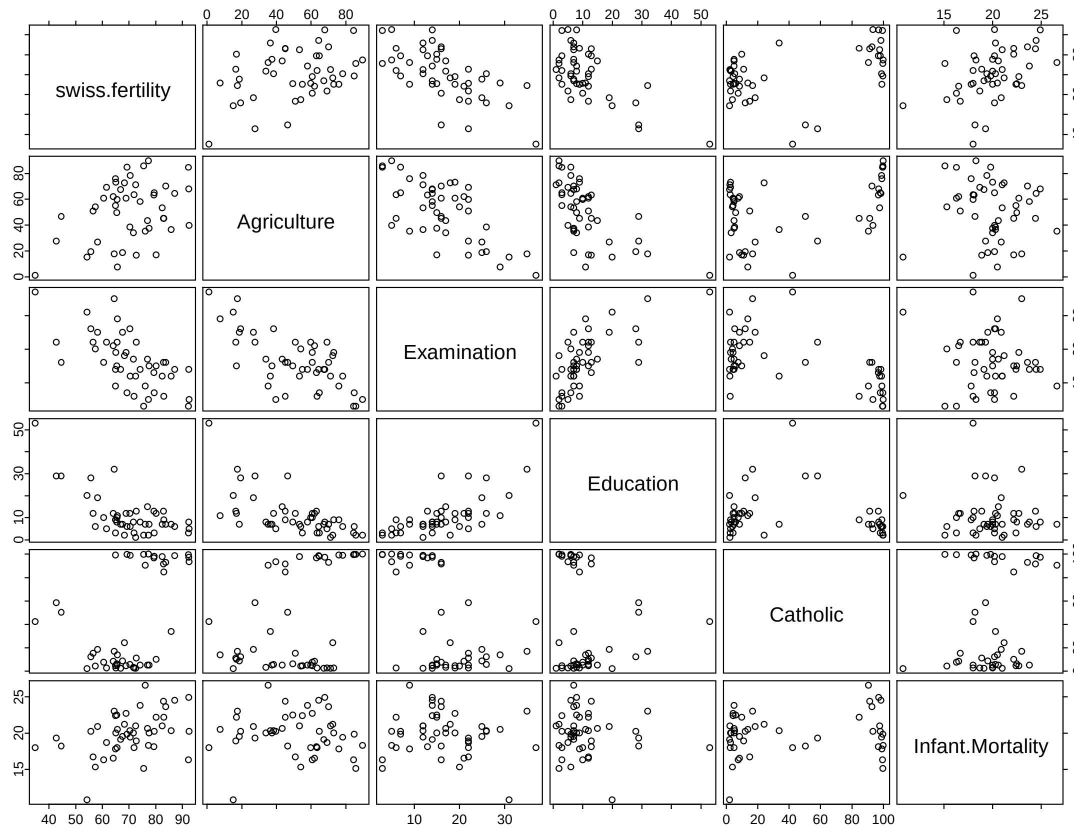
```
 swiss.fertility    Agriculture    Examination     Education       Catholic
   Min.:35.00       Min.: 1.20     Min.: 3.00     Min.: 1.00     Min.:  2.20
 1st Qu.:64.70    1st Qu.:35.90   1st Qu.:12.00  1st Qu.: 6.00  1st Qu.:  5.20
  Median:70.40     Median:54.10    Median:16.00   Median: 8.00   Median: 15.10
   Mean:70.14       Mean:50.66      Mean:16.49     Mean:10.98     Mean: 41.14
 3rd Qu.:78.45    3rd Qu.:67.65   3rd Qu.:22.00  3rd Qu.:12.00  3rd Qu.: 93.15
   Max.:92.50       Max.:89.70      Max.:37.00     Max.:53.00     Max.:100.00
 Infant.Mortality
   Min.:10.80
 1st Qu.:18.15
  Median:20.00
   Mean:19.94
 3rd Qu.:21.70
Max.:26.60
```

Q1.c.

```
> cor(swiss.df)
                 swiss.fertility Agriculture Examination   Education   Catholic
swiss.fertility        1.0000000  0.35307918  -0.6458827 -0.66378886  0.4638833
    Agriculture        0.3530792  1.00000000  -0.6865422 -0.63952252  0.4011097
    Examination       -0.6458827 -0.68654221   1.0000000  0.69841530 -0.5728530
      Education       -0.6637889 -0.63952252   0.6984153  1.00000000 -0.1540185
       Catholic        0.4638833  0.40110968  -0.5728530 -0.15401847  1.0000000
Infant.Mortality       0.4165560 -0.06085861  -0.1140216 -0.09932185  0.1755114
                 Infant.Mortality
swiss.fertility        0.41655603
    Agriculture       -0.06085861
    Examination       -0.11402160
      Education       -0.09932185
       Catholic        0.17551138
Infant.Mortality       1.00000000
```

Q1.d.

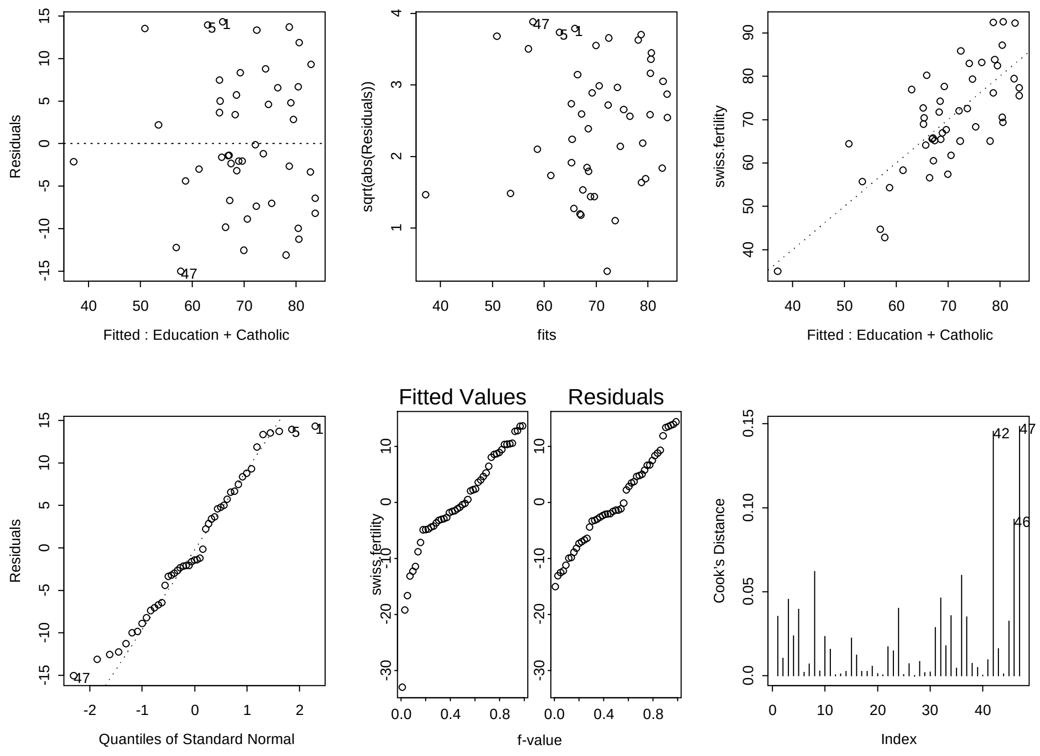
```
> pairs(swiss.df)
```



Q1.e.

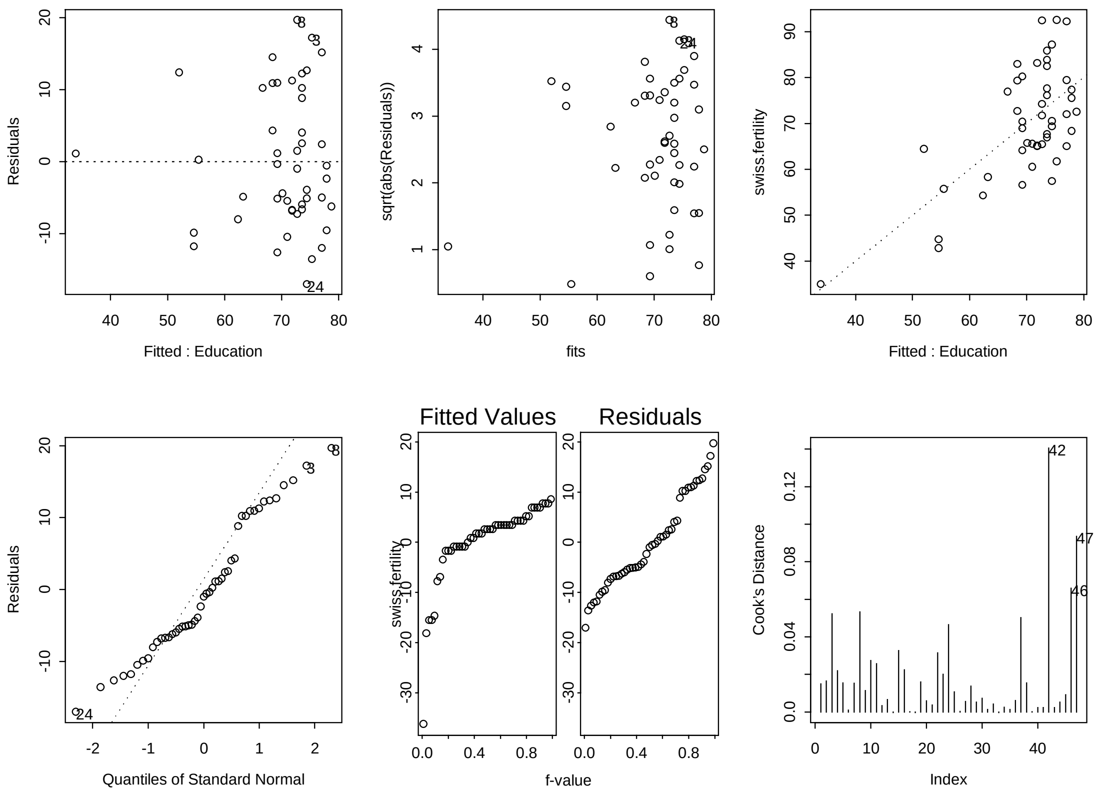
```
> fit1 <- lm(swiss.fertility~Education+Catholic)
> summary(fit1)
Call: lm(formula = swiss.fertility ~ Education + Catholic)
Residuals:
    Min     1Q Median    3Q   Max
 -15.04 -6.576 -1.426 6.126 14.32

Coefficients:
               Value Std. Error  t value Pr(>|t|)
(Intercept)  74.2319   2.3518    31.5636   0.0000
  Education  -0.7882   0.1293    -6.0968   0.0000
   Catholic   0.1109   0.0298     3.7224   0.0006

Residual standard error: 8.331 on 44 degrees of freedom
Multiple R-Squared: 0.5746
F-statistic: 29.71 on 2 and 44 degrees of freedom, the p-value is 6.822e-009

Correlation of Coefficients:
          (Intercept) Education
Education -0.6838
 Catholic -0.6144      0.1540
```

```
> par(mfrow=c(2,3))
> plot(fit1)
```



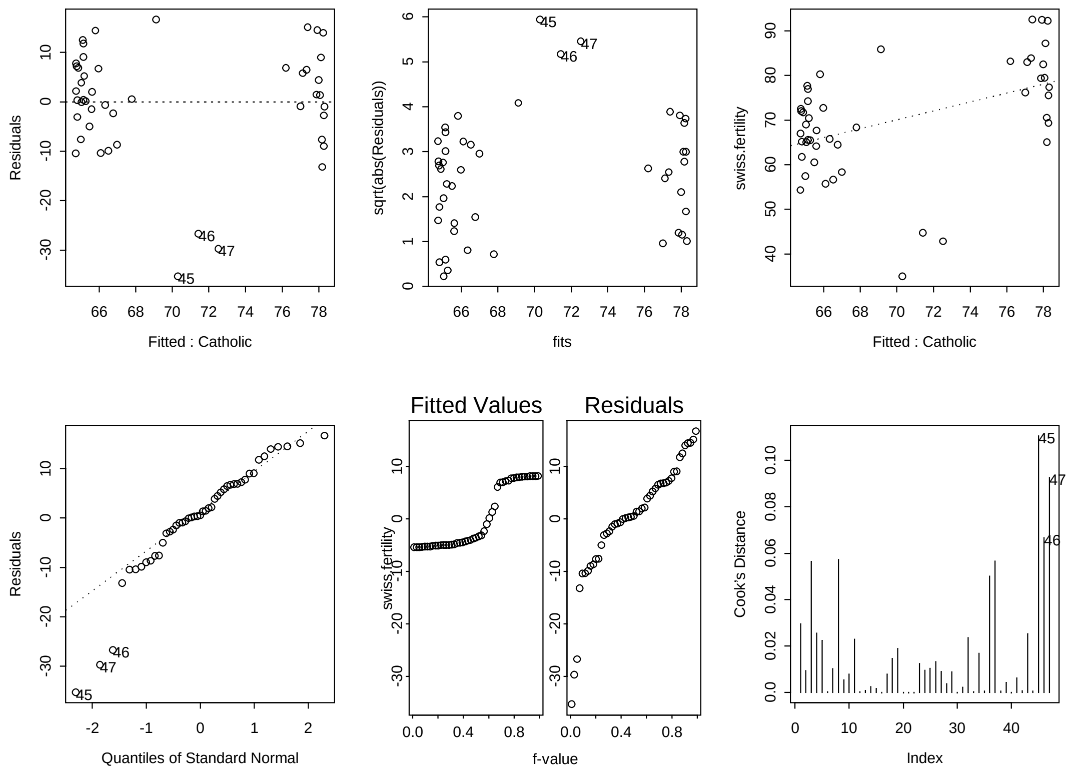
```
> fit2 <- update(fit1,.~.-Catholic)
> summary(fit2)
Call: lm(formula = swiss.fertility ~ Education)
Residuals:
    Min     1Q Median    3Q   Max
 -17.04 -6.711 -1.011 9.526 19.69

Coefficients:
              Value Std. Error  t value Pr(>|t|)
(Intercept)  79.6101   2.1041    37.8357   0.0000
  Education  -0.8624   0.1448    -5.9536   0.0000

Residual standard error: 9.446 on 45 degrees of freedom
Multiple R-Squared: 0.4406
F-statistic: 35.45 on 1 and 45 degrees of freedom, the p-value is 3.659e-007

Correlation of Coefficients:
          (Intercept)

Education -0.7558
```

```
> par(mfrow=c(2,3))

> plot(fit2)
```



```
> fit3 <- update(fit1,.~.-Education)
> summary(fit3)
Call: lm(formula = swiss.fertility ~ Catholic)
Residuals:
   Min     1Q Median    3Q   Max
 -35.3 -4.056 0.5114 6.854 16.68

Coefficients:
              Value Std. Error t value Pr(>|t|)
(Intercept) 64.4266  2.3047    27.9549  0.0000
   Catholic  0.1389  0.0395     3.5126  0.0010

Residual standard error: 11.19 on 45 degrees of freedom
Multiple R-Squared: 0.2152
F-statistic: 12.34 on 1 and 45 degrees of freedom, the p-value is 0.001023

Correlation of Coefficients:
         (Intercept)
Catholic -0.7061
```

```
> par(mfrow=c(2,3))

> plot(fit3)
```



Q1.h.

```
> anova(fit1,fit2)
Analysis of Variance Table

Response: swiss.fertility

                Terms Resid. Df      RSS      Test Df Sum of Sq F Value
1 Education + Catholic        44 3053.627
2            Education        45 4015.236 -Catholic -1 -961.6083 13.8559
        Pr(F)
1
2 0.0005575191
> anova(fit1,fit3)
Analysis of Variance Table

Response: swiss.fertility

                Terms Resid. Df      RSS       Test Df Sum of Sq  F Value
1 Education + Catholic        44 3053.627
2             Catholic        45 5633.347 -Education -1  -2579.72 37.17142
         Pr(F)
1
2 2.431488e-007
```

Prefer the model that has the two.

Solution: Functions

Q1.

```
> myplot.fun <- function(x){
+     hist(x)
+     }
```

Q2.

```
> fix(myplot.fun)
function(x)
{
    par(mfrow = c(1, 2))
    hist(x)
    boxplot(ozone)
}
```

Q3.

```
function(x = 0:100, type = "l")
{
   if(any(x < 0)) {
         warning("Negative values are not allowed")
   }
   else {
         plot(x, sqrt(x), type = type)
   }
}
```

Q4.

```
> fix(plot.any)
function(func,min.x,max.x,num.x=100,type="l",...)
{
  x <- seq(min.x,max.x,length=100)
  y <- func(x)
  plot(x,y,type=type,...)
}
```

Q5.

```
> coef.var <- function(x){
+   cv <- sqrt(var(x))/mean(x)
+   cv
+   }
```

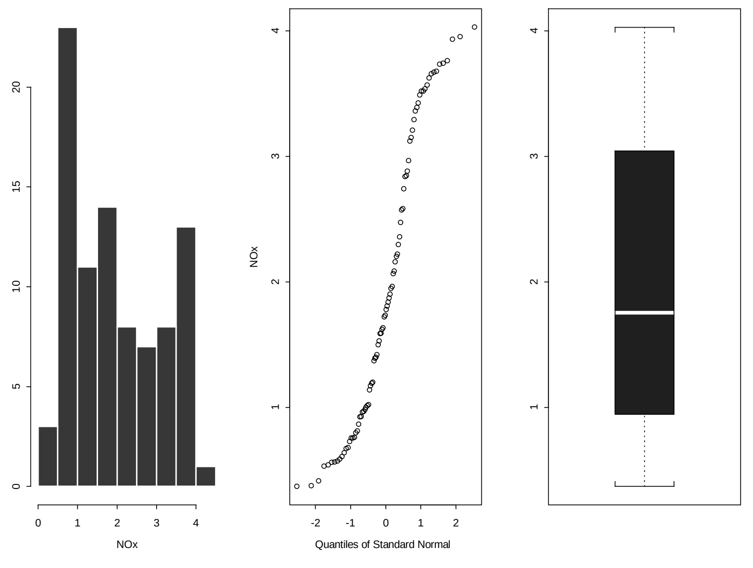
Q6.

```
> fix(coef.var)
function(x, na.rm = F)
{
    if(na.rm) {
            x <- x[!is.na(x)]
    }
    cv <- sqrt(var(x))/mean(x)
    cv
}
```

Solution: Graphic

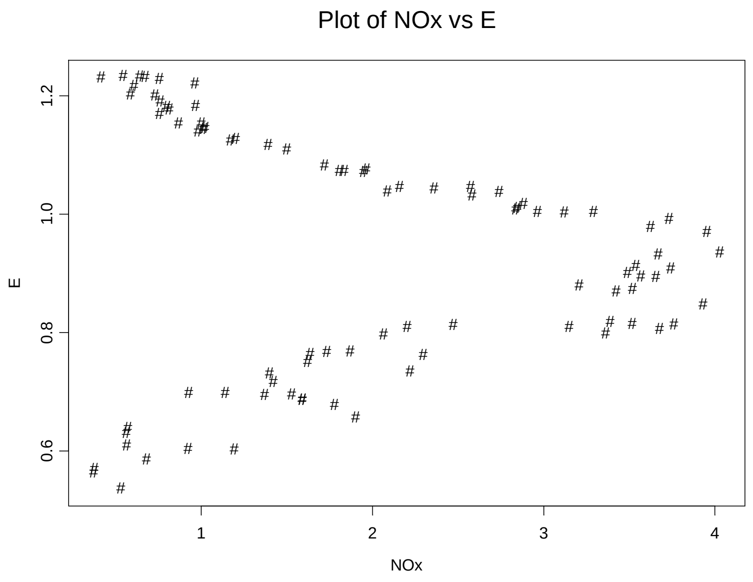
Q1. a.

```
> attach(ethanol)
> par(mfrow=c(1,3))
> hist(NOx)
> qqnorm(NOx)
> boxplot(NOx)
```



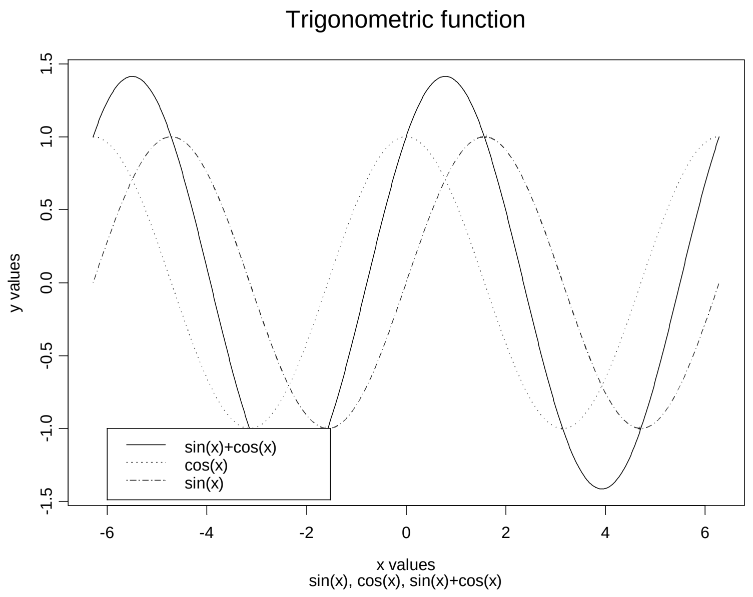
Q1.b.

```
> plot(NOx,E,pch="#")
> title("Plot of NOx vs E")
```



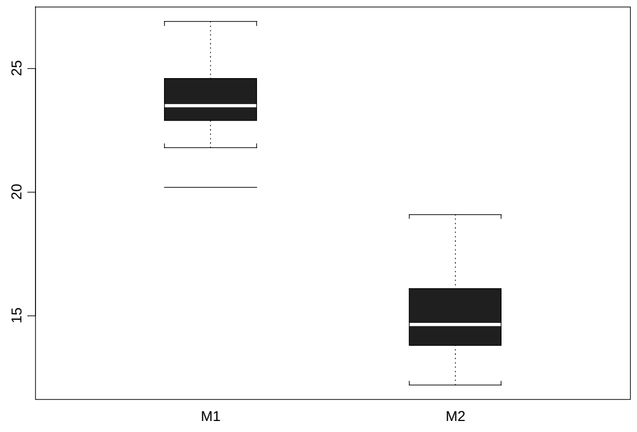
Q2.

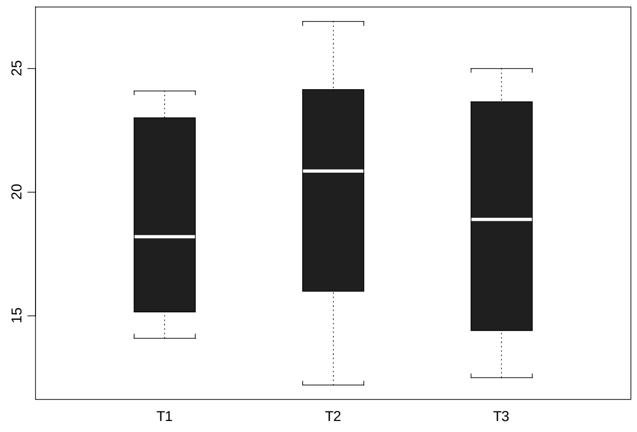
```
> x <- seq(-2*pi,2*pi,length=1000)
> plot(x,sin(x)+cos(x),type="l",xlab="x values",ylab="y values")
> lines(x,cos(x),lty=2,col=2)
> lines(x,sin(x),lty=3,col=3)
> title(main="Trigonometric function",
+       sub="sin(x), cos(x), sin(x)+cos(x)")
> legend(-6,-1,c("sin(x)+cos(x)","cos(x)","sin(x)"),lty=1:3,col=1:3)
```



Q3.

```
> boxplot(split(gun$Rounds,gun$Method))
> boxplot(split(gun$Rounds,gun$Team))
```



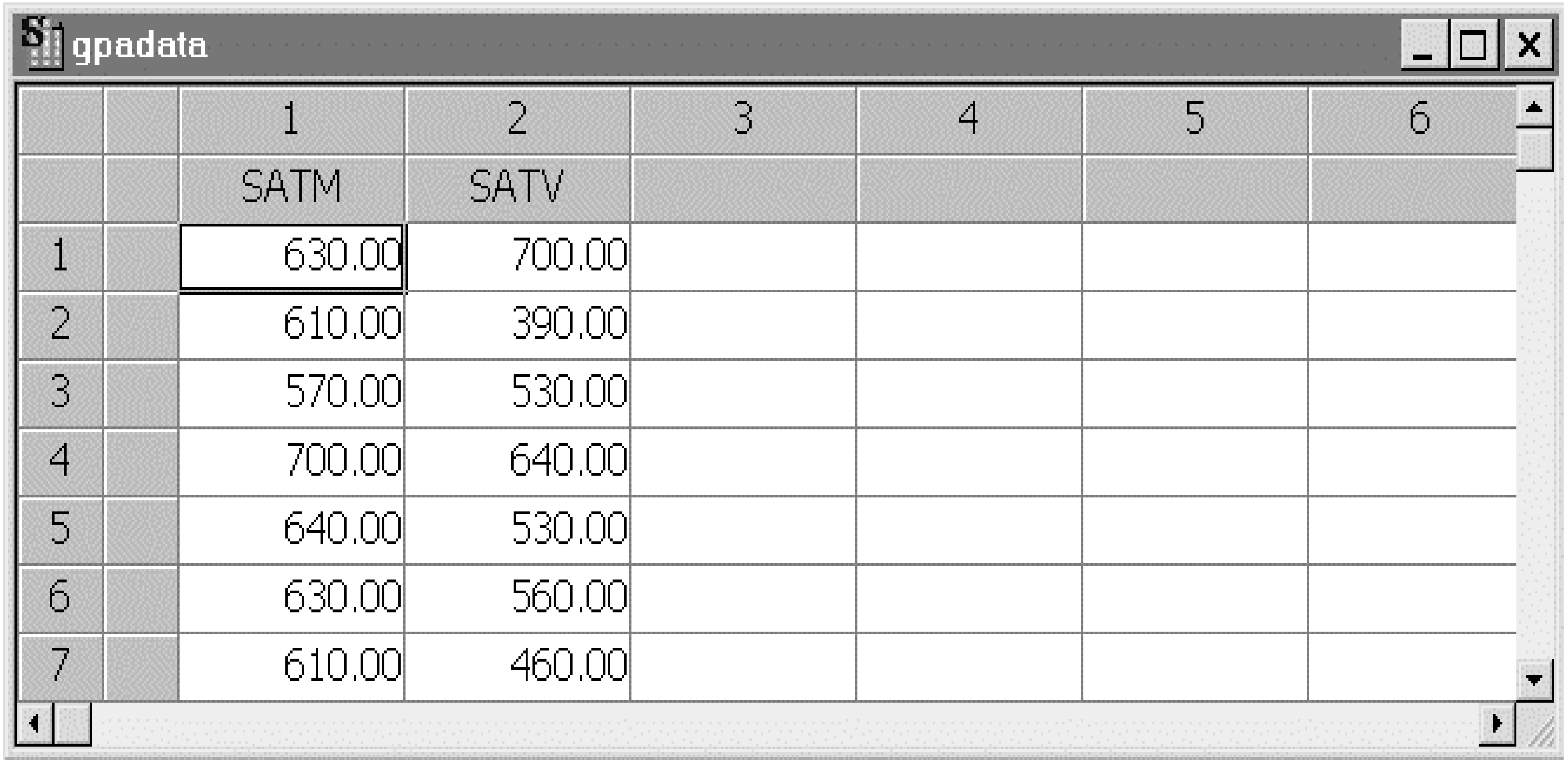


Solution: Data Import & Export

Q1. In S-Plus do the following

```
> import.data("c:\\datasets\\Gpadata.sd2",
+   FileType="SAS",
+   NameRow="1",
+   DataFrame="gpadata")
```

- The result is

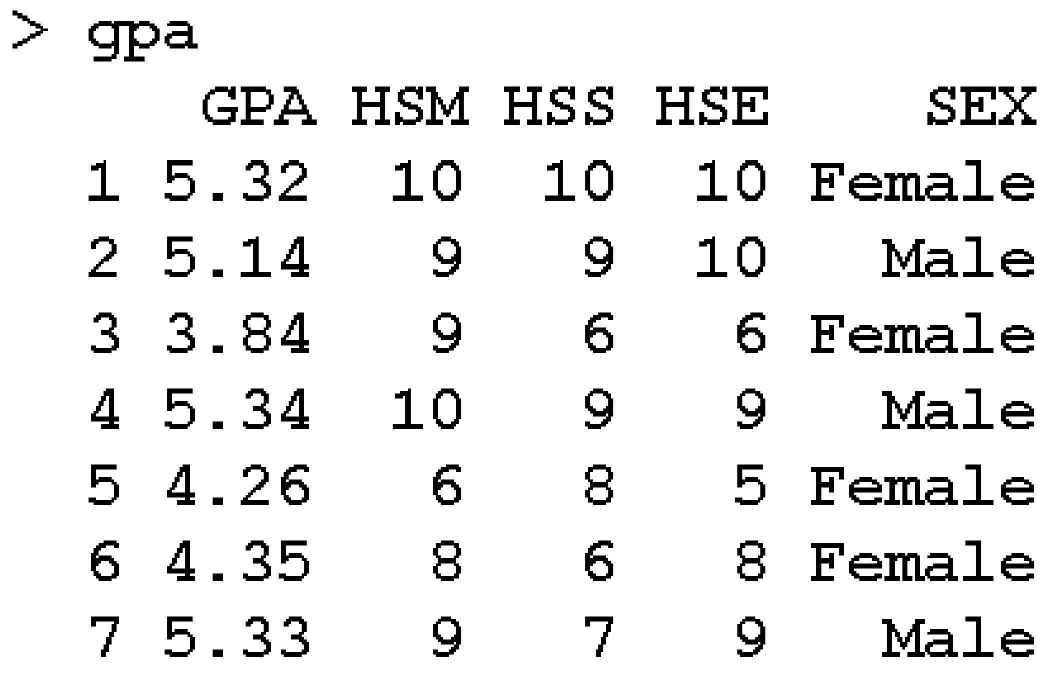


		1	2	3	4	5	6
		SATM	SATV				
1		630.00	700.00				
2		610.00	390.00				
3		570.00	530.00				
4		700.00	640.00				
5		640.00	530.00				
6		630.00	560.00				
7		610.00	460.00				

Q2. In SPLUS do the following:

```
> gpa <- read.table("c:\\datasets\\gpa.txt",
+                   header=T,sep=",")
```

- The result is



```
> gpa
   GPA HSM HSS HSE    SEX
1 5.32  10  10  10 Female
2 5.14   9   9  10   Male
3 3.84   9   6   6 Female
4 5.34  10   9   9   Male
5 4.26   6   8   5 Female
6 4.35   8   6   8 Female
7 5.33   9   7   9   Male
```

Q3. In SPLUS do the following:

```
> export.data(DataSet="gas",
+             Delimiter=",",
+             ColumnNames=T,
+             Quotes=T,
+             FileType="ASCII",
+             FileName="c:\\datasets\\gas.txt")
```

Q4. In SPLUS do the following:

```
> export.data(DataSet="gas",
+             FileType="SAS",
+             FileName="c:\\datasets\\gas.sd2")
```

Q5. In SPLUS do the following:

```
>data.dump(c("gun","car.all"),"c:\\datasets\\datall.dmp")
```

Q6. In SPLUS do the following:

```
> sink("c:\\datasets\\galaxy.txt")
> galaxy
> sink()
```

Q7. Do it in class